



(12) 发明专利

(10) 授权公告号 CN 110827385 B

(45) 授权公告日 2024. 07. 09

(21) 申请号 201910492554.0

(22) 申请日 2019.06.06

(65) 同一申请的已公布的文献号

申请公布号 CN 110827385 A

(43) 申请公布日 2020.02.21

(30) 优先权数据

16/101,180 2018.08.10 US

(73) 专利权人 辉达公司

地址 美国加利福尼亚州

(72) 发明人 S·莱内 T·艾拉 T·卡拉斯

G·穆特乐 W·P·小组霍尔

R·C·小巴比奇 C·科尔布

I·利亚马斯

(74) 专利代理机构 北京市磐华律师事务所

11336

专利代理师 高伟 姜晓丹

(51) Int.Cl.

G06T 15/04 (2011.01)

G06T 15/60 (2006.01)

G06T 1/20 (2006.01)

(56) 对比文件

US 2011316855 A1, 2011.12.29

审查员 朱佳彧

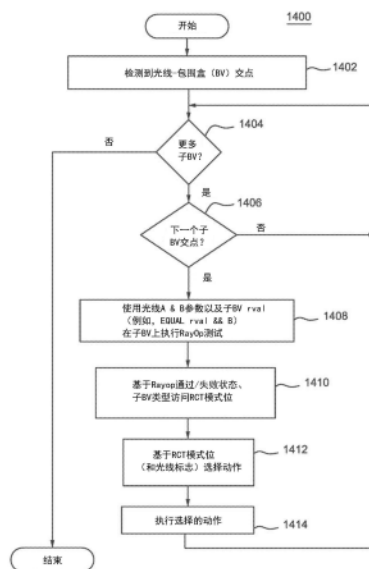
权利要求书3页 说明书52页 附图35页

(54) 发明名称

树遍历的特定于查询的行为修改

(57) 摘要

本发明公开了一种树遍历的特定于查询的行为修改。在一些示例中描述了用于以高度动态的特定于查询的方式改变加速数据结构的遍历的方法和系统,其中每个查询指定测试参数、测试操作码以及测试结果到动作的映射。在示例性光线跟踪实现中,执行光线对层次包围盒的遍历,其中根据使用测试操作码和在光线数据结构中指定的测试参数以及在层次包围盒的节点中指定的另一测试参数执行的测试的结果来改变遍历的默认行为。在示例性实现中,遍历协处理器被配置为执行层次包围盒的遍历。



1. 一种光线追踪方法,包括:

访问对应于光线的光线数据和对应于场景的加速数据结构,其中被访问的光线数据包括:(a) 光线参数;(b) 操作码;以及(c) 一个或更多个模式标志;

根据所述光线数据遍历所述加速数据结构,其包括:基于所述光线的几何属性和所述加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行(a) 相交测试;以及执行(b) 辅助光线操作测试,其包括:

基于所述操作码,将所述至少一个节点的节点参数与所述光线参数或源自所述光线参数的值进行比较;以及

至少基于所述比较的结果,从所述一个或更多个模式标志中选择至少一个模式标志;

所述遍历进一步包括:基于所述相交测试和所述辅助光线操作测试的结果,基于所选的至少一个模式标志执行改变所述遍历的状态或选择和与所述光线相交的所述加速数据结构的一个或更多个节点相对应的输出数据中的至少一个;以及

将来自所述遍历的输出数据提供到渲染过程,以用于渲染所述场景的至少一部分。

2. 根据权利要求1所述的方法,其中,执行相交测试和辅助光线操作测试包括基于所述操作码选择所述辅助光线操作测试。

3. 根据权利要求2所述的方法,其中,执行相交测试和辅助光线操作测试还包括基于每个来自所述光线数据和所述至少一个节点的至少一个参数执行所选择的光线操作测试。

4. 根据权利要求3所述的方法,其中来自所述光线数据的所述至少一个参数是每光线可配置的,并且来自所述至少一个节点的所述至少一个参数是每节点可配置的。

5. 根据权利要求3所述的方法,其中,执行相交测试和辅助光线操作测试还包括:基于所述光线的非几何属性和所述加速数据结构来执行所选择的光线操作测试。

6. 根据权利要求2所述的方法,其中,执行相交测试和辅助光线操作测试还包括:基于来自所述至少一个节点和所述操作码的一个参数来执行所选择的光线操作测试。

7. 根据权利要求2所述的方法,其中,执行改变遍历的状态或选择输出数据中的至少一个包括:至少基于在所述光线数据中规定的一个或更多个模式标志选择改变遍历的状态和选择输出数据中的至少一个。

8. 根据权利要求2所述的方法,其中,基于所述至少一个节点的节点类型来选择所述一个或更多个模式标志。

9. 根据权利要求2所述的方法,其中,执行改变遍历的状态或选择输出数据中的至少一个还包括:基于所述一个或更多个模式标志,改变在所述遍历期间遍历所述加速数据结构中的各个节点的顺序。

10. 根据权利要求2所述的方法,其中,执行改变遍历的状态或选择输出数据中的至少一个还包括:基于所述一个或更多个模式标志,通过剔除所述加速数据结构的一部分动态地改变所述加速数据结构的结构。

11. 根据权利要求2所述的方法,其中,基于与所述光线相关联的几何属性和与所述加速数据结构相关联的几何属性对所述加速数据结构中的至少一个节点执行(a) 相交测试;以及(b) 执行辅助光线操作测试进一步包括:基于所述操作码执行所述辅助光线操作测试,其中所述操作码提供用于比较由与所述光线相关联的一个几何属性缩放的在所述相交测试期间所计算的值,并且由与所述光线相关联的另一几何属性偏置到与所述至少一个节点

相关联的至少一个几何参数。

12. 根据权利要求2所述的方法, 其中, 基于所述光线的几何属性和所述加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行 (a) 相交测试; 以及执行 (b) 辅助光线操作测试进一步包括: 基于所述操作码执行所述辅助光线操作测试, 其中所述操作码提供用于将与所述光线相关联的距离与所述至少一个节点的至少一个参数进行比较。

13. 根据权利要求2所述的方法, 其中, 基于所述光线的几何属性和所述加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行 (a) 相交测试; 以及执行 (b) 辅助光线操作测试包括: 基于所述操作码执行所述辅助光线操作测试, 其中所述操作码用于将基于所述光线数据中的至少一个参数的第一值与基于所述至少一个节点的至少一个参数的第二值进行比较。

14. 根据权利要求2所述的方法, 其中, 基于所述光线的几何属性和所述加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行 (a) 相交测试; 以及执行 (b) 辅助光线操作测试进一步包括: 基于所述操作码执行所述辅助光线操作测试, 其中所述操作码用于将基于来自所述至少一个节点的至少一个参数的在所述至少一个节点中表示的表面区域的大小与沿着光线发生相交之处的距离进行比较。

15. 根据权利要求1所述的方法, 还包括: 至少部分地基于由执行所述遍历的遍历处理器提供的输出数据, 由并行处理器将所述场景渲染到显示器。

16. 一种光线追踪方法, 包括:

访问对应于光线的光线数据和对应于场景的加速数据结构;

根据所述光线数据遍历所述加速数据结构, 其包括: 基于所述光线的几何属性和所述加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行 (a) 相交测试; 以及执行 (b) 辅助光线操作测试;

所述遍历进一步包括: 基于所述相交测试和所述辅助光线操作测试的结果, 执行改变遍历的状态或选择和与所述光线相交的所述加速数据结构的一个或多个节点相对应的输出数据中的至少一个, 其中基于在所述光线数据中指定的一个或多个模式标志来选择改变遍历的状态和选择输出数据中的至少一个还包括: 根据所述至少一个节点中的一个或多个覆写标志进行选择; 以及

将来自所述遍历的输出数据提供到渲染过程, 以用于渲染所述场景的至少一部分。

17. 根据权利要求16所述的方法, 其中, 所述至少一个节点是实例节点。

18. 根据权利要求16所述的方法, 其中, 所述至少一个节点表示图元, 并且所述覆写标志中的至少一个被配置为覆写在所述光线数据中规定的剔除配置。

19. 一种包括遍历协处理器的系统, 所述遍历协处理器包括用于遍历加速数据结构的硬件处理电路, 被配置为:

访问对应于光线的光线数据和对应于场景的加速数据结构, 其中被访问的光线数据包括: (a) 光线参数; (b) 操作码; 以及 (c) 一个或多个模式标志;

根据所述光线数据遍历所述加速数据结构, 其包括: 基于所述光线的几何属性和加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行 (a) 相交测试; 以及执行 (b) 辅助光线操作测试, 其包括:

基于所述操作码, 将所述至少一个节点的节点参数与所述光线参数或源自所述光线参

数的值进行比较;以及

至少基于所述比较的结果,从所述一个或更多个模式标志中选择至少一个模式标志;所述遍历进一步包括:基于所述相交测试和所述辅助光线操作测试的结果,基于所选的至少一个模式标志执行改变所述遍历的状态或选择和与所述光线相交的所述加速数据结构的一个或更多个节点相对应的输出数据中的至少一个;以及

将来自所述遍历的输出数据提供到渲染过程,以用于渲染所述场景的至少一部分。

20.根据权利要求19所述的系统,还包括:并行处理器,被配置为生成与所述光线相对应的所述光线数据和与所述场景的模型相对应的所述加速数据结构。

21.一种用于加速遍历树形数据结构的协处理器,包括:

接口电路,被配置为与多处理器和存储器通信以接收多个查询数据结构、所述树形数据结构和查询命令,其中每个查询数据结构包括查询测试参数、测试操作码和模式标志,并且其中所述树形数据结构中的节点包括节点测试参数;以及

树遍历电路,配置为:

响应于所述查询命令,根据每个所述查询数据结构遍历所述树形数据结构;以及

针对每个所述查询数据结构确定:

基于所述操作码和各个查询数据结构的查询测试参数以及所述树形数据结构中的节点的节点测试参数的测试结果;以及

基于所述结果和所述各个查询数据结构的所述模式标志改变所述遍历或返回的一个或更多个输出。

22.一种光线追踪方法,包括:

访问对应于光线的光线数据和对应于场景的加速数据结构,

根据所述光线数据遍历所述加速数据结构,其包括:基于所述光线的几何属性和所述加速数据结构的几何属性对所述加速数据结构中的至少一个节点执行(a)相交测试;以及执行(b)辅助光线操作测试,

其中包括在所述光线数据中的操作码用于执行以下项中的至少一者:

将由与所述光线相关联的一个几何属性缩放的并且由与所述光线相关联的另一几何属性偏置的在所述相交测试期间所计算的值与和所述至少一个节点相关联的至少一个几何参数进行比较;

将与所述光线相关联的距离与所述至少一个节点的至少一个参数进行比较;

将基于所述光线数据中的至少一个参数的第一值与基于所述至少一个节点的至少一个参数的第二值进行比较;或

将至少基于来自所述至少一个节点的至少一个参数的在所述至少一个节点中表示的表面区域的大小与沿所述光线相交发生之处的距离进行比较;

在所述遍历期间,基于所述相交测试和所述辅助光线操作测试的结果,执行改变所述遍历的状态或选择和与所述光线相交的所述加速数据结构的一个或更多个节点相对应的输出数据中的至少一个;以及

将来自所述遍历的输出数据提供到渲染过程,以用于渲染所述场景的至少一部分。

树遍历的特定于查询的行为修改

[0001] 相关申请的交叉引用

[0002] 本申请涉及以下共同转让的美国专利和专利申请,其全部内容通过引用并入本文:2014年12月8日提交的标题为“树状数据结构的短堆栈遍历(Short Stack Traversal of Tree Data Structures)”的美国申请No.14/563,872;标题为“基于块的层次包围盒(Block-Based Bounding Volume Hierarchy)”的美国专利No.9,582,607;标题为“用于基于块的层次包围盒的相对编码(Relative Encoding For A Block-Based Bounding Volume Hierarchy)”的美国专利No.9,552,664;2015年3月18日提交的标题为“光束追踪(Beam Tracing)”的美国专利No.9,569,559;标题为“基于多个局部坐标系的树状数据结构(Tree Data Structures Based on a Plurality of Local Coordinate Systems)”的美国专利No.10,025,879;2015年6月11日提交的标题为“基于块的几何数据无损压缩(Block-Based Lossless Compression of Geometric Data)”的美国申请No.14/737,343;以及以下与其同时提交的美国申请:

[0003] • (卷号:6610-0032/18-AU-0127)标题为“在没有着色器干预下对交点进行连续层次包围盒遍历的方法(Method for Continued Bounding Volume Hierarchy Traversal On Intersection Without Shader Intervention)”;

[0004] • (卷号:6610-0033/18-AU-0128),标题为“用于数据路径调度的高速缓存请求的有效分组方法(Method for Efficient Grouping of Cache Requests for Datapath Scheduling)”;

[0005] • (卷号:6610-0034/18-SC-0144)标题为“鲁棒且高效的多处理器-协处理器接口(Query-Specific Behavioral Modification of Tree Traversal)”;

[0006] • (卷号:6610-0036/18-SC-0145)标题为“保守严密的光线三角形相交(Conservative Watertight Ray Triangle Intersection)”;

[0007] • (卷号:6610-0037/18-SC-0149)标题为“处理无序不透明和 α 光线/图元交点的方法(Method for Handling Out-of-Order Opaque and A Ray/Primitive Intersections)”;

[0008] • (卷号:6610-0039/18-AU-0170)标题为“硬件中树遍历机制的前进和可编程超时的方法(Method for Forward Progress and Programmable Timeouts of Tree Traversal Mechanisms in Hardware)”。

技术领域

[0009] 本技术涉及计算机图形,更具体地涉及光线追踪。更具体地,该技术涉及计算机图形处理的硬件加速,包括但不限于光线追踪。更具体地,本文的示例非限制性技术涉及基于硬件的遍历协处理器,其有效地遍历加速数据结构,例如,用于实时光线追踪以及该协处理器的遍历行为的特定于查询的行为修改。

[0010] 背景和发明内容

[0011] 如果你在你之前环顾视觉场景,你会注意到你看到的一些最有趣的视觉效果是由

光线与表面相互作用产生的。这是因为光是我们看到的唯一东西。我们看不到对象-我们看到的是对象反射或折射的光。我们可以看到的大多数对象都反射光(对象的颜色由对象反射哪部分光和吸收哪部分光确定)。闪亮的表面,诸如金属表面、光泽表面、陶瓷、液体表面和其他各种对象(甚至是人眼的角膜)都可以作为镜面反射光的镜子。例如,闪亮的金属表面将以其碰撞(hit)表面时相同的角度反射光。对象还可以通过防止光线到达相对于光源为对象后面的其他表面来投射阴影。如果你环顾四周,你会注意到反射的数量和种类以及阴影的数量、种类和长度取决于许多因素,包括场景中光的数量和类型。单个点光源(例如单个遥远的灯泡)将产生单次反射和硬阴影。面光源(诸如窗户或光板)产生不同种类的反射高光和更柔和的阴影。多个光通常会产生多个反射和更复杂的阴影(例如,三个分离的点光源将产生三个阴影,这些阴影可以根据光相对于对象的位置而重叠)。

[0012] 如果你在调查场景时移动你的头部,你会注意到反射在位置和形状上发生变化(阴影也一样)。通过改变你的视点,你可以改变你的眼睛检测到的光线的不同角度。这种情况瞬间发生-你移动你的头部则立即改变视觉场景。

[0013] 喝一杯茶的简单行为是复杂的视觉体验。在你反射房间内的每个光之前,桌子上有光泽的陶瓷杯的各个光泽表面,杯子为每个光投下阴影。杯子中茶的移动表面本身是反光的。你可以看到茶叶表面上的光的小反射图像,甚至是茶叶表面部分的较小反射,在该处液体向上弯曲以与杯壁相交。杯壁还将阴影投射到杯子中的液体表面上。将杯子举到你的嘴处会导致这些反射和阴影随着你的视点的变化而移动和闪烁,并且随着液体表面因运动而变得波动而移动和闪烁。

[0014] 我们认为这些复杂的反射和阴影是理所当然的。我们的大脑擅长解码阴影和反射的位置、大小和形状,并将它们用作视觉线索。这部分地是我们如何辨别对象相对于彼此的位置、我们如何区分一个对象与另一个对象以及我们如何了解对象的构成。不同的对象表面反射不同。硬金属的镜面(镜子类型)反射创建反射对象的图像,而粗糙表面的漫反射负责颜色并以更柔和的方式照亮对象。根据光照的类型,阴影可以是柔和的、漫射的或硬的和不同的,阴影的长度和方向将取决于光线相对于对象和我们眼睛的角度。

[0015] 初学艺术家通常不会尝试显示反射或阴影。他们倾向于绘制没有阴影、没有反射或高光的平面场景。过去的计算机图形也是如此。

[0016] 在过去的30年中,实时计算机图形已经取得了巨大的进步。随着20世纪80年代功能强大的图形处理单元(GPU)的发展,通过提供3D硬件图形管线,实时响应于用户输入基于纹理映射的多边形图元生成3D图形显示成为可能。这种实时图形处理器建立在称为扫描变换光栅化的技术之上,该技术是从单个点或透视图确定可见性的手段。使用这种方法,三维对象从几何图元构成的表面建模,通常是多边形(诸如三角形)。扫描变换过程建立图元多边形顶点并将其投影到视图平面上,并填充图元边缘内部的点。参见例如Foley, Van Dam, Hughes等,计算机图形:原理与实践(Computer Graphics: Principles and Practice)(第2版, Addison-Wesley 1995和第三版, Addison-Wesley 2014)。

[0017] 长期以来,硬件一直用于确定每个多边形表面应如何着色和纹理映射,以及光栅化着色的、纹理映射的多边形表面以供显示。典型的三维场景通常由数百万个多边形构成。快速的现代GPU硬件可以实时响应于用户输入有效地处理用于每个显示帧(每1/30或1/60秒)的数百万个图元。得到的图形显示已经用于各种实时图形用户界面,包括但不限于增强

现实、虚拟现实、视频游戏和医学成像。但传统上,这种交互式图形硬件尚无法准确地建模和描绘反射和阴影。

[0018] 一些已经在该基本扫描变换光栅化方法上构建了其他技术,以允许实时图形系统在渲染阴影和反射时实现一定量的真实感。例如,纹理映射有时用于模拟3D场景中的反射和阴影。通常完成其的一种方法是从不同视角变换、投影和光栅化对象,将光栅化结果写入纹理映射,并对纹理映射进行采样以提供反射映射、环境映射和阴影。虽然这些技术已被证明是有用且适度成功的,但它们并不是在所有情况下都能很好地工作。例如,所谓的“环境映射”通常可能需要假设环境距离对象无限远。另外,环境映射对象通常可能无法反射自身。参见例如http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter07.html。这些限制的结果是因为传统的计算机图形硬件-虽然足够快以实现出色的多边形渲染-但却无法执行精确逼真的反射和阴影所需的光可视化。有些人将反射和阴影的光栅/纹理近似比作AM收音机的视觉等效物。

[0019] 存在另一种图形技术,其确实执行了用于反射和阴影的物理上真实的可见性确定。它被称为“光线追踪”。光线追踪是在20世纪60年代末开发出来的,并在20世纪80年代得到了改进。参见例如Apple,“一些用于固体的着色机器渲染的技术 (Some Techniques for Shading Machine Renderings of Solids)” (SJCC 1968) 第27-45页;Whitted,“用于着色显示的改进的照明模型 (An Improved Illumination Model for Shaded Display)” 第343-349页,ACM通讯第23卷第6期(1980年6月);以及Kajiya,“渲染方程 (The Rendering Equation)”,计算机图形学 (SIGGRAPH 1986会议记录,第20卷,第143-150页)。从那时起,光线追踪已经用于非实时图形应用,例如设计和电影制作。任何看过“多莉去哪儿 (Finding Dory)” (2016) 或其他皮克斯动画电影的人都看到了计算机图形的光线追踪方法的结果-即逼真的阴影和反射。参见例如Hery等人,“在皮克斯处迈向双向路径追踪 (Towards Bidirectional Path Tracing at Pixar)” (2016)。

[0020] 光线追踪是在各种渲染算法中使用的图元,包括例如路径追踪和梅特波利斯 (Metropolis) 光照传输。在示例算法中,光线追踪通过对穿过场景的光传输进行建模以使用射线光学计算所有全局效果 (包括例如来自闪亮表面的反射) 来模拟光的物理学。在光线追踪的这种使用中,当光线从可能多个光源到视点穿过三维场景时,可以尝试追踪数百或数千个光线中的每一个。通常,这些光线穿过场景相对于眼睛进行追踪,并针对场景中所有几何形状的数据库进行测试。光线可以从光向前追踪到眼睛,或从眼睛向后追踪到光,或者可以追踪它们以查看从虚拟相机开始以及从眼睛开始的路径是否具有清晰的视线。该测试确定最近的交点 (以便确定眼睛可见的是什么) 或者从对象表面朝向光源追踪光线以确定空间中是否存在阻止光传输到该点的任何干预。因为光线与现实中的光线相似,所以它们提供了许多真实的效果,使用过去三十年来实现的基于光栅的实时3D图形技术是不可能实现该效果的。因为来自场景内每个光源的每个照射光线在穿过场景中的每个对象时被评估,所以得到的图像看起来好像是在现实中拍摄的。因此,这些光线追踪方法长期以来一直用于专业图形应用,例如设计和电影,其中它们已经成为基于光栅的渲染的主导。

[0021] 光线追踪的主要挑战通常是速度。光线追踪要求图形系统针对每个帧计算和分析入射到构成场景的每个表面 (并且可能由其反射) 的数百万条光线中的每一条。过去,这种大量的计算复杂性是不可能实时执行的。

[0022] 现代GPU 3D图形管线在渲染着色的纹理映射表面处如此快速的一个原因是它们有效地使用了相干性。在传统的扫描变换中,假设所有一切都通过公共图像平面中的公共窗口观察并向下投射到单个有利点。每个三角形或其他图元通过图形管线发送并覆写一些像素。可以为从该三角形渲染的所有像素共享所有相关计算。因此,对应于穿过窗口的相干视线的像素的矩形图块可以对应于在同一流式处理器中以锁步方式运行的线程组。假设落在三角形边缘之间的所有像素被假设是运行相同着色器的相同材料,并从相同纹理获取相邻的纹素组。相反,在光线追踪中,光线可以在公共点(光源或虚拟相机镜头)处开始或结束,但是当它们在场景中传播并与不同材料相互作用时,它们很快发散。例如,每条光线执行搜索以找到最近的对象。可以执行一些高速缓存和结果共享,但由于每条光线可能会碰到不同的对象,因此GPU传统上利用的、与纹理映射、着色三角形有关的相干性类型不存在(例如,不存在一个共同的有利点、窗口和图像平面用于光线追踪)。这使得光线追踪在计算上比其他图形方法更具挑战性-因此在交互的基础上执行起来要困难得多。

[0023] 已经进行了许多研究以使得追踪光线的过程更加有效和及时。参见例如,Glassner, 光线追踪介绍(Ray Tracing Introduction)(Academic Press Inc.,1989)。因为光线追踪中的每条光线本质上独立于其余光线进行评估,所以光线追踪被称为“令人尴尬地平行”。参见例如Akenine-Möller 等人,实时渲染在章节9.8.2,412页(第三版.CRC出版社2008)。如上所述,光线追踪涉及针对场景中的所有对象和表面有效地测试每条光线。称为“加速数据结构”和相关过程的优化允许图形系统在加速数据结构上使用“分而治之(divide-and-conquer)”的方法来建立光线碰撞的表面以及光线未碰撞的表面。每条光线以个性化的方式遍历加速数据结构。这意味着将更多处理器专用于光线追踪可以提供近乎线性的性能提升。随着图形处理系统的并行性的增加,一些人开始设想可以实时执行光线追踪的可能性。例如,2000年中期在萨尔州大学的工作产生了一个早期的用于交互式光线追踪的专用硬件系统,其为使用几何着色器、顶点着色器和光照着色器提供了一定程度的可编程性。参见Woop等人的“RPU:用于实时光线追踪的可编程光线处理单元(RPU:A Programmable Ray Processing Unit for Real Time Ray Tracing)”(ACM 2005)。作为另一示例,先进的渲染技术基于源自ARM1的AR250/350渲染处理器阵列开发了“RenderDrive(渲染器驱动)”,并通过自定义管线进行增强,以用于光线/三角形相交和SIMD向量和纹理数学,但没有固定功能遍历逻辑。参见例如http://www.graphicshardware.org/previous/www_2001/presentations/Hot3D_Daniel_Hall.pdf。

[0024] 然后,在2010年,NVIDIA利用NVIDIA GPU和其他高度并行架构的高并行度来开发OptiX™光线追踪引擎。参见Parker等人的“OptiX:通用光线追踪引擎(OptiX:A General Purpose Ray Tracing Engine)”(ACM Transactions on Graphics,Vol.29,No.4,Article 66,July 2010)。除了API(应用程序编程接口)的改进之外,OptiX™提供的一项改进是改进用于查找光线和场景几何形状之间的交点的加速数据结构。这种加速数据结构通常是由光线追踪遍历算法使用的空间或对象分层,以有效地搜索可能与给定光线相交的图元。OptiX™提供了许多不同的加速结构类型,可供应用程序选择。节点图中的每个加速结构可以是不同的类型,允许高质量静态结构与动态更新的静态结构的组合。

[0025] OptiX™可编程光线追踪管线提供了显着的进步,但是通常仍然无法在相对便宜的计算平台上为复杂的3D场景提供对用户输入的实时交互响应。从那时起,NVIDIA一直在开

发用于光线追踪的硬件加速能力。参见例如US9,582,607;US 9,569,559;US20160070820以及US20160070767。

[0026] 鉴于用于响应于例如用户输入渲染任意复杂度的高质量图像的真实交互式实时光线追踪图形处理系统的巨大潜力,进一步的工作是可能的并且是期望的。

附图说明

[0027] 图1示出了示例性非限制性光线追踪图形系统。

[0028] 图2A示出了示例性镜面对象。

[0029] 图2B示出了包围盒(Bounding Volume)内的示例性对象。

[0030] 图2C示出了图2B的包围盒的示例性盒细分。

[0031] 图2D、图2E和图2F示出了包围盒的盒细分的示例性进一步水平,以创建层次包围盒(BVH)。

[0032] 图2G示出了由图元表面组成的对象的示例部分,在这种情况下是三角形。

[0033] 图3A-图3C示出了用于确定光线是否穿过包含几何形状的包围盒以及光线是否与几何形状相交的示例性简化光线追踪测试。

[0034] 图4示出了示例性光线追踪流程图。

[0035] 图5A-图5C示出了示例性不同的光线-图元相交场景。

[0036] 图6A和图6B示出了纹理映射如何影响光线-图元相交结果的示例。

[0037] 图7A和图7B示出了光线实例变换。

[0038] 图8A示出了示例性非限制性层次包围盒(BVH)。

[0039] 图8B示出了图形或树状的示例性加速数据结构。

[0040] 图9示出了包括树遍历单元(TTU)的简化示例性非限制遍历协处理器。

[0041] 图10A示出了示例性非限制性光线追踪着色管线流程图。

[0042] 图10B和图10C示出了更详细的光线追踪管线。

[0043] 图11A示出了根据一些实施例的示例性查询数据结构。

[0044] 图11B示出了根据一些实施例的BVH中的示例性节点数据结构。

[0045] 图12示出了根据一些实施例的示例性遍历堆栈数据结构。

[0046] 图13示出了根据一些实施例的示例性结果队列数据结构。

[0047] 图14示出了根据一些实施例的示出在遍历期间针对BVH中的包围盒执行的辅助光线操作测试的过程。

[0048] 图15示出了根据一些示例性实施例的示出当叶节点相交时图14的过程的过程。

[0049] 图16示出了根据一些示例性实施例的示出当光线操作测试结果被传递到光线-图元测试块时图14的过程的过程。

[0050] 图17示出了示例性并行处理单元(PPU)。

[0051] 图18示出了示例性存储器分区单元。

[0052] 图19示出了图17的并行处理单元内的示例通用处理集群(GPC)。

[0053] 图20和图21示出了示例性流式多处理器。

[0054] 图22是使用图17的PPU实现的处理系统的概念图。

[0055] 图23展开图22以示出另外的互连设备。

[0056] 图24是由图19的GPC实现的图形处理管道的概念图。

具体实施方式

[0057] 本文的技术提供硬件能力,其将光线追踪加速到这样的程度,即它将光线追踪的能力带到游戏和其他交互式实时计算机图形,最初在阴影和反射中实现高效质量并最终实现全局照明。在实践中,这意味着通过相同图形渲染系统上的软件可能达到的数量级的因子或更高的因子来加速光线追踪。

[0058] 更详细地,示例性非限制性技术提供了用于加速光线追踪的专用硬件。在非限制性实施例中,硬件协处理器(本文称为“遍历协处理器”或在一些实施例中称为“树遍历单元”或“TTU”)加速支持交互式光线追踪的某些过程,包括光线-包围盒相交测试、光线-图元相交测试和光线“实例”变换。

[0059] 在一些非限制性实施例中,遍历协处理器对加速数据结构执行查询,以用于在潜在大规模并行流式多处理器(SM)上运行的进程。遍历协处理器遍历加速数据结构以发现关于给定光线如何与加速数据结构描述或表示的对象交互的信息。对于光线追踪,与例如在运行不同类型的线程(例如,顶点线程和像素线程)的逻辑管线阶段之间执行一次操作的固定功能单元相比,遍历协处理器是可调用的。

[0060] 在一些非限制性实施例中,加速数据结构包括递归地封装越来越小的包围盒细分的包围盒的层次(层次包围盒或BVH)。最大容量的包围盒可以称为“根节点”。包围盒的这种层次的最小细分(“叶节点”)包含项目(item)。项目可以是定义对象表面的图元(例如,诸如三角形的多边形)。或者,项目可以是球体,其包含作为项目存在的全新级别的世界,因为它尚未添加到BVH(想想猫身上的衣领魅力来自“黑衣人”,其中包含它里面的整个微型星系)。如果项目包括图元,则遍历协处理器针对图元测试光线,以确定与光线相交的对象表面以及沿着光线可见的对象表面。

[0061] 遍历协处理器针对宽范围的包围盒执行每条光线的测试,并且可以剔除不与该光线相交的任何包围盒。从界定场景中所有一切的根节点开始,遍历协处理器针对较小(可能重叠)的子包围盒测试每条光线,这又界定了BVH的后代分支。光线跟随光线碰撞其他节点的包围盒的子指针,直到达到BVH的叶或终端节点(盒)。一旦遍历协处理器遍历加速数据结构以到达包含几何图元的终端或“叶”节点,它就执行加速的光线-图元相交测试,其确定光线是否与该图元相交(因此与图元定义的对象表面相交)。光线图元测试可以提供有关与光线相交的图元的附加信息,其可用于确定着色和可视化所需的表面的材料属性。通过加速数据结构的递归遍历使得遍历协处理器能够发现与光线相交的所有对象图元,或者与光线相交的最接近的(从视点的角度看)图元(在某些情况下,它是沿着光线的视点唯一可见的图元)。

[0062] 遍历协处理器还加速每条光线从世界空间到对象空间的变换,以获得图元的越来越精细的包围盒封装,并减少场景上的那些图元的复制。在场景中不同位置、方向和比例下多次复制的对象可以在场景中表示为实例节点,其将世界空间BVH中的包围盒和叶节点与可以应用于世界空间光线的变换相关联,以将其变换为对象坐标空间,以及表示为指向对象空间BVH的指针。这避免了在世界空间中多次复制对象空间BVH数据,从而节省了存储器和相关的存储器访问。实例变换通过将光线变换为对象空间而不是要求将几何形状或层次

包围盒变换为世界(光线)空间来提高效率,并且还与图形处理执行的以使图元可视化的附加的传统光栅化过程兼容。

[0063] 因此,目前公开的某些非限制性实施例提供了遍历协处理器,3D图形处理管线中的一个或一组流式多处理器SM的新子单元。为了理解遍历协处理器在整体图片中适合的位置,理解大多数或所有现代光线追踪器所采用的算法的一些基本原理可能会有所帮助。但是应该指出的是,本文的技术提供了一种通用能力,用于为在GPU中运行的线程确定沿着指定方向从给定点开始的最近的可见事物,或者两个点之间是否有任何东西。这种能力的一个常见用例是在开始追踪来自已经在三角形上光栅化的点的光线的过程中使用传统扫描变换技术。所公开的技术可以但不一定取代或替代扫描变换技术,并且通常可以增强它并且与扫描变换技术结合使用以增强图像的照片般真实的反射、阴影和其他效果。

[0064] 光线追踪技术

[0065] 通常,光线追踪是一种渲染方法,其中光线用于确定场景中各种元素的可见性。光线追踪可用于确定沿光线是否有任何对象可见(例如,测试几何图元上的阴影点与光源上的点之间的遮挡物),并且还可用于评估反射(例如,可以涉及执行遍历以确定沿着视线的最近的可见表面,使得在流式处理器上运行的软件可以评估与被碰撞的内容相对应的材料着色功能-这反过来可以根据相交的对象的材料特性发射一条或更多条附加光线到场景中),以确定沿着光线返回眼睛的光。在经典的Whitted式光线追踪中,光线从视点通过像素网格射入场景,但其他路径遍历也是可能的。通常,针对每条光线,找到最近的对象。然后通过将光线从其射到场景中的每个光源并且发现其间是否有任何对象来确定该交点被照亮或处于阴影中。不透明对象阻挡光线,而透明对象则会使光线衰减。其他光线可以从交点产生。例如,如果相交表面是有光泽的或镜面的,则在反射方向上产生光线。光线可以接受相交的第一个对象的颜色,而其又测试阴影的交点。递归地重复该反射过程,直到达到递归限制或后续反弹的潜在贡献低于阈值。也可以在透明固体对象的折射方向上产生光线,并再次递归地评估。参见上文引用的Akenine-Möller等人。因此,光线追踪技术允许图形系统开发物理上正确的反射和阴影,其不遭受扫描变换技术的限制和伪影。

[0066] 遍历协处理器

[0067] 遍历协处理器执行的基本任务是针对场景中的所有图元(在一个实施例中通常是三角形)测试光线,并报告最接近的碰撞(根据沿光线测量的距离)或简单地报告遇到的第一(不一定最近)碰撞,这取决于用例。简捷算法将是 $O(n)$ 强力搜索。然而,通过预先处理场景几何形状并提前构建合适的加速数据结构,可以将平均情况复杂度降低到 $O(\log n)$ 。在光线追踪中,当使用加速数据结构时,找到光线最接近(或针对阴影任何)交点的时间通常是 n 个对象的阶数 $O(\log n)$ 。例如,通常用于现代光线追踪加速数据结构的类型的层次包围盒(BVH)通常具有 $O(\log n)$ 搜索行为。

[0068] 层次包围盒

[0069] 现代光线追踪器最常使用的加速数据结构是包括嵌入的轴对齐包围盒(AABB)的层次包围盒(BVH)。BVH的叶节点包含要相交测试的图元(例如,三角形)。BVH通常由图形或树形结构数据表示来表示。在这种情况下,遍历协处理器可以称为“树遍历单元”或“TTU”。

[0070] 给定BVH,光线追踪相当于树搜索,其中光线所访问的树中的每个节点具有用于每个后代分支或叶的包围盒,并且光线仅访问其相应包围盒与光线相交的后代分支或叶。通

过这种方式,只有少数图元必须明确地进行相交测试,即那些驻留在与光线相交的叶节点中的图元。在示例性非限制性实施例中,遍历协处理器加速树遍历(包括光线-盒测试)和光线-图元测试两者。作为遍历的一部分,遍历协处理器还可以处理“实例变换”-将光线从世界空间坐标系变换为实例网格(对象空间)的坐标系,例如,以便避免将图元顶点变换到世界空间的计算复杂性。它可以以MIMD(多指令,多数据)方式实现,这意味着在遍历协处理器内部一次独立地处理光线。

[0071] 示例的非限制性实时交互式光线追踪系统

[0072] 图1示出了用于使用场景或一个或多个对象的三维(3D)数据生成图像的示例的实时光线交互式追踪图形系统100。系统100包括输入设备110、一个或多个处理器120、一个或多个图形处理单元(GPU)130、存储器140和一个或多个显示器150。图1中所示的系统可以采用任何形状因素,包括但不限于个人计算机、智能手机或其他智能设备、视频游戏系统、可穿戴虚拟或增强现实系统、基于云的计算系统、车载图形系统,片上系统(SoC)等。

[0073] 处理器120可以是多核中央处理单元(CPU),其可操作以实时交互式响应于输入设备110而执行应用程序,其输出包括用于在显示器150上显示的图像。显示器150可以是任何种类的显示器,例如固定显示器、头戴式显示器(诸如显示器眼镜或护目镜)、其他类型的可穿戴显示器、手持显示器、车载显示器等。例如,处理器120可以基于从输入设备110(例如,操纵杆、惯性传感器、环境光传感器等)接收的输入执行应用程序,并指示GPU 130生成示出应用程序进展的图像以在显示器150上显示。

[0074] 基于处理器120上的应用程序的执行,处理器可以发出用于GPU 130的指令,以使用存储在存储器140中的3D数据生成图像。GPU 130包括用于实时加速图像生成的专用硬件。例如,GPU 130能够实时处理用于数千或数百万个图元(多边形)的信息,这是由于GPU能够比传统软件-驱动的CPU更快地执行重复和高度并行的专用计算任务,例如多边形扫描变换。例如,与处理器120不同,其可具有多个核心,具有可一次处理少量软件线程的大量高速缓存存储器,GPU 130可包括数百或数千个处理核心或并行运行的“流式多处理器”(SM) 132。

[0075] 在一个示例性实施例中,GPU 130包括多个可编程流式多处理器(SM) 132,以及包括图元引擎134和光栅引擎136的基于硬件的图形管线。GPU 130的这些组件被配置为使用称为“扫描变换光栅化”的技术执行实时图像渲染,以在二维显示器150上显示三维场景。在光栅化中,3D场景的几何构建块(例如,点、线、三角形、四边形、网格等)被映射到显示器的像素(通常通过帧缓冲存储器)。

[0076] GPU 130将3D模型的几何构建块(即,诸如三角形之类的多边形图元)变换为2D图像的像素,并为每个像素分配初始颜色值。图形管线可以通过定义或调整像素的颜色值来将着色、透明度、纹理和/或颜色效果应用于图像的部分。最终像素值可以经过抗锯齿、过滤并提供给显示器150以供显示。多年来,许多软件和硬件的改进使用光栅化技术以一个或多个显示器150上的高显示分辨率(例如4096x 2160像素或更高)在实时图形(即每秒30至60帧)所需的帧速率下提高了主观图像质量。

[0077] 遍历协处理器添加到架构

[0078] 为了使GPU 130能够以有效的方式实时地执行光线追踪,GPU被提供有耦合到一个或多个SM 132的遍历协处理器138。遍历协处理器138包括被配置为执行通常在光线追踪

算法使用的操作的硬件组件。遍历协处理器138的目标是将光线追踪中使用的操作加速到这样的程度,即它将光线追踪的能力带到实时图形应用(例如,游戏),从而实现高质量的阴影、反射和全局照明。在一些示例性实施例中,遍历协处理器138包括特定于查询的遍历硬件139,其使得遍历协处理器的行为的特定于查询的编程能够例如增加光线追踪操作对正被渲染的场景中的动态变化等的灵活性和响应性。如下面更详细讨论的,遍历协处理器138的结果可以与GPU 130中执行的其他图形相关操作一起使用或作为其替代。

[0079] 在所示的示例性架构中,称为“遍历协处理器”138的新硬件组件用于加速某些任务,包括但不限于光线追踪。光线追踪是指将光线投射到场景中,并确定该光线是否以及在哪儿与场景的几何形状相交。这种基本的光线追踪可见性测试是计算机图形学中作为各种渲染算法和技术基础的基础图元。例如,光线追踪可以与光栅化和z缓冲一起使用或作为光栅化和z缓冲的替代,用于采样场景几何形状。它还可以用作环境映射和阴影纹理的替代(或与之结合),以产生比通过纹理技术或其他光栅“黑客”实现的更逼真的反射、折射和阴影效果。为了克服可以通过光栅化实现的图像质量的限制,系统100还可以使用光线追踪技术生成整个图像或图像的部分。光线追踪也可以用作基本图元,以精确地模拟基于物理的渲染算法中的光传输,例如路径追踪、光子映射、Metropolis光传输和其他光传输算法。

[0080] 更具体地,SM 132和遍历协处理器138可以协作,以将光线投射到3D模型中并确定该光线是否以及在何处与模型的几何形状相交。光线追踪直接模拟穿过虚拟环境或场景的光。光线相交的结果与表面纹理、观察方向和/或光照条件一起用于确定像素颜色值。由与遍历协处理器138一起工作的SM 132执行的光线追踪允许计算机生成的图像以与现实世界的照片或视频无法区分的方式捕获阴影、反射和折射。由于光线追踪技术部分地由于需要追踪的大量光线而比光栅化更加计算密集,因此遍历协处理器138能够在硬件中加速该过程的某些计算密集度更高的方面。

[0081] 在本文的示例非限制性技术中,遍历协处理器138加速光线盒测试和光线图元测试两者。作为遍历的一部分,它还可以处理至少一级实例变换,将光线从世界空间坐标系变换为实例网格的坐标系。在示例非限制性实施例中,遍历协处理器138以MIMD方式完成所有这些操作,这意味着在遍历协处理器内部独立地处理光线一次。

[0082] 在示例非限制性实施例中,遍历协处理器138作为SM(流式多处理器)132的服务方(协处理器)操作。换句话说,在示例非限制性实施例中,遍历协处理器138不独立地操作,而是遵循SM 132的命令以比SM 132自身执行更有效地执行某些计算密集的光线追踪相关任务。

[0083] 在所示的示例中,遍历协处理器138通过SM 132指令接收命令并将结果写回SM寄存器文件。对于许多常见用例(例如,具有至多一级实例化的不透明三角形),遍历协处理器138可以服务于光线追踪查询而无需与SM 132进一步交互。更复杂的查询(例如,除了三角形或多级实例化之外涉及经过 α 测试的三角形、图元),可能需要多次往返。除了追踪光线之外,遍历协处理器138还能够执行更一般的空间查询,其中AABB或两个AABB(我们称之为“波束”)之间的挤压盒(extruded volume)取代光线。因此,虽然遍历协处理器138特别适合于加速与光线追踪相关的任务,但它也可用于执行除光线追踪之外的任务。

[0084] 除了遍历协处理器138之外,用于支持图1的系统100的示例非限制性技术还为多个单元提供了额外的加速光线追踪增强以及用于BVH构建的大量努力。BVH构建不需要是硬

件加速的(尽管在一些非限制性实施例中可以是),而是可以使用在SM 132和/或CPU 120和/或其他开发系统上(例如,在应用程序开发期间)运行的高度优化的软件例程来实现。以下说明描述了遍历协处理器138的软件可见行为、与周围单元(SM 132和存储器子系统)的接口以及作为完整光线追踪解决方案的一部分的附加特征,例如对SM 132组和存储器高速缓存系统的某些增强等。

[0085] 如上所述,遍历协处理器138允许快速遍历加速数据结构(例如,BVH)以确定数据结构中哪些图元(例如,用于生成场景的三角形)与查询数据结构(例如,光线)相交。示例性实施例通过提供用于遍历过程的特定于查询的编程的技术和系统,使得能够进一步改进,诸如,基于特定查询(例如,光线)特性和/或相交节点特性来选择遍历行为的能力,改进场景中的动态变化的响应性,改进BVH遍历速度等。

[0086] 遍历加速数据结构

[0087] 加速光线追踪的好方法是使用加速数据结构。加速数据结构以有助于快速确定特定光线很可能与之相交的对象的哪个部分以及快速地拒绝光线不会与之相交的场景的大部分的方式表示对象或场景的3D模型。层次包围盒(BVH)数据结构是一种类型的加速数据结构,其可以帮助减少要测试相交的数量。BVH数据结构表示具有包围盒的场景或对象,并将包围盒细分为越来越小的包围盒,其终止于包含几何图元的叶节点。包围盒是分层的,这意味着最高级别(level)包含它下面的级别,该级别包含它下面的下一级别,依此类推。在一个实施例中,叶节点可能潜在地与层次包围盒中的其他叶节点重叠。

[0088] 为了说明层次包围盒如何工作,图2A-图2G示出了递归地细分为越来越小层次的包围盒的茶壶。图2A示出了茶壶对象,图2B示出了包围整个茶壶的包围盒202(在这种情况下是盒子、立方体或长方体)。可以通过其顶点有效地限定的包围盒202提供对象的空间位置的指示,并且通常尺寸略大于对象。

[0089] 加速结构构造的第一阶段获取所引用几何形状的包围盒。这是通过对对象中的每个几何图元执行包围盒程序来实现的,该包围盒程序返回其输入图元的保守轴对齐包围盒,例如图2B中所示的盒202。使用这些包围盒作为加速结构的基本图元,提供了针对任意用户定义的几何形状(包括单个结构内的几种类型的几何形状)追踪光线的必要抽象。因为在图2B中,包围盒202大于并且完全包含茶壶,不与包围盒相交的光线不能与茶壶相交,尽管与包围盒相交的光线可能或可能不与茶壶相交。因为包围盒202容易由其在3D空间中的顶点的x,y,z坐标定义,并且光线由其在3D空间中的x,y,z坐标定义,所以用于确定光线是否与包围盒202相交的光线-包围盒测试是简单的(尽管可以使用一些变换来调整到不同的坐标系,如下面将解释的)。

[0090] 图2C示出了细分为较小的内含(contained)包围盒的包围盒202。虽然为了说明的目的而在此示出的细分方案是所谓的8进制细分或“八叉树”,其中每个盒被细分为八个统一尺寸的较小盒,许多其他空间层次和细分方案是已知的,例如二进制树、四进制树、k-d树、二进制空间分区(BSP)树和层次包围盒(BVH)树。参见例如USP 9,582,607。

[0091] 图2C中所示的每个细分的包围盒可以进一步细分。图2D示出了图2C的细分盒204中的一个被进一步细分以提供额外细分的封装的包围盒。如图2D所示,一些细分的包围盒包括茶壶的部分而一些不包括。不包含茶壶的部分的盒不会进一步细分,因为进一步的细分不提供关于茶壶的进一步空间信息。已经细分的包含茶壶的至少一部分的包围盒可以进

一步递归地细分-就像Seuss博士的帽子里的猫回来啦(1958年)的帽子里出现的一连串越来越小的猫一样。包含几何形状的包围盒202内的空间的部分被递归地细分,以允许遍历协处理器138使用盒细分来有效地发现几何形状相对于任何给定光线的位置。可以注意到,虽然盒的空间细分或主动细分是可能的,但许多实现将创建提前定义盒和子盒的层次结构。在这种情况下,构建器通常可以从各个三角形向上构建层次,而不是从整个场景向下构建。向上构建意味着你不需要确定某个细分的盒是否包含任何内容,因为根据定义它包含在盒细分层次中位于其下面的内容。

[0092] 图2E示出了另外的这样的包围盒204细分为另一个较小的内含包围盒206,该包围盒206在该示例中仅包含茶壶的壶嘴加上茶壶的壁上的另一个表面,并且图2F示出了将包围盒206附加细分为更小的内含细分208,该细分208封装茶壶的壶嘴的末端。取决于构建BVH的方式,可以根据需要不断更进一步细分包围盒208-并且遍历协处理器138使图1系统100能够有效地将BVH向下遍历到任何任意细分级别。递归细分的数量和配置将取决于被建模的3D对象的复杂性和配置以及其他因素,例如期望的分辨率、对象距视点的距离等。

[0093] 在某些细分级别(对于BVH的不同部分可以是不同级别),遍历协处理器138遇到构成被建模的封装对象的几何形状。使用树的类比,连续的盒细分是树干、分枝、树枝和细枝,并且最终在树的尖端即叶上显示几何形状。在这种情况下,图2G显示了由几何图元的示例性网格定义的茶壶壶嘴的表面。所示的几何图元是三角形,但是可以使用其他几何图元,例如四边形、线条、矩形、二次曲面、补丁或熟悉现有技术的人所知的其他几何图元(在一个实施例中,这样的其他类型的图元可以表示为或变换为三角形)。网格中的几何图元表示被建模的对象的3D表面的形状。此处示出的示例是网格,但是所包围的几何形状可以包括不连续的几何形状,例如可能未连接的粒子。在示例性非限制性实施例中,遍历协处理器138还利用该几何形状加速光线相交测试,以快速确定任何给定光线碰撞哪个三角形。确定光线-图元相交包含将每个图元的顶点的空间xyz坐标与光线的xyz坐标进行比较,以确定图元定义的光线和表面是否占据相同的空间。光线-图元相交测试可能是计算密集型的,因为可能要测试许多三角形。例如,在图2G所示的网格中,仅茶壶的壶嘴由超过一百个三角形组成-尽管在一些实现方式中可以更有效地进一步进行盒细分,从而限制在任何此类“叶节点”到类似16或更少的东西的三角形的数量。

[0094] 如上所述,光线追踪程序确定场景的什么几何图元与光线相交。然而,由于3D场景中的大量图元,测试每个几何图元的相交可能不是有效或可行的。加速数据结构(例如BVH)允许快速确定哪些包围盒可以被忽略,哪些包围盒可以包含相交的几何图元,以及哪些相交的几何图元对于可视化而言是重要的而哪些不是。

[0095] 光线相交测试

[0096] 图3A-图3C示出了应用于包括三角形网格320的图2G包围盒208的光线追踪。图3A示出了包括包围盒310和315的虚拟空间中的光线302。为了确定光线302在网格320中是否与一个或多个三角形相交,每个三角形可以直接针对光线302进行测试。但是为了加速该过程(因为该对象可以包含数千个三角形),首先针对包围盒310和315测试光线302。如果光线302不与包围盒相交,则它不与包围盒内部的任何三角形相交,并且可以针对该光线忽略包围盒内部的所有三角形。因为在图3A中,光线302未碰撞(miss)了包围盒310,所以不需要对该包围盒内的网格320的三角形进行相交测试。虽然包围盒315与光线302相交,但是包围

盒315不包含任何几何形状,因此不需要进一步的测试。

[0097] 另一方面,如果光线(诸如图3B中所示的光线304)与包含几何形状的包围盒310相交,则光线可能与或可能不与包围盒内的几何形状相交,因此需要对几何形状本身执行进一步的测试以找到可能的相交。因为图3B和图3C中的光线304、306与包含几何形状的包围盒310相交,所以需要执行进一步的测试以确定包围盒内部的任何(以及哪些)图元是否相交。在图3B中,对与图元的相交的进一步测试将指示即使光线304穿过包围盒310,它也不与包围盒所包围的任何图元相交(或者,如上所述,包围盒310可以进一步进行盒细分,以便可以使用包围盒相交测试来揭示光线不与任何几何形状相交,或者更具体地,光线可以与哪些图元相交)。

[0098] 图3C示出了包围盒310与光线306相交并且包含与光线306相交的几何形状的情况。遍历协处理器138测试光线306和各个图元之间的相交,以确定光线与哪些图元相交。

[0099] 光线追踪操作

[0100] 图4是概述遍历协处理器138如上所述与一个或更多个SM 132协作执行的示例性光线追踪操作的流程图。图4的操作由遍历协处理器138和与其交互的SM 132协作执行。因此,遍历协处理器138可以从SM 132接收光线的标识,并且遍历状态列举光线必须穿过的一个或更多个BVH中的一个或更多个节点。遍历协处理器138确定光线与BVH数据结构的哪些包围盒(“光线-complet”测试512)相交,并随后确定光线是否与相交的包围盒中的一个或更多个图元相交以及与哪些三角形相交(“光线-图元测试”520)。在示例性非限制性实施例中,“complets”(压缩小树)指定层次包围盒的根节点或内部节点(即,盒),其具有子项(child),子项是其他complet或每个complet的单个类型的叶节点。

[0101] 首先,遍历协处理器138检查光线的遍历状态。如果遍历协处理器138保持用于光线的堆栈为空,则遍历完成。如果堆栈顶部存在条目,则遍历协处理器138向存储器子系统发出请求以检索该节点。然后,遍历协处理器138执行包围盒测试512以确定BVH数据结构的包围盒是否与SM 132指定的特定光线相交(步骤512、514)。如果包围盒测试确定包围盒未与光线相交(步骤514中为“否”),则不需要对可视化执行任何进一步测试,并且遍历协处理器138可以将该结果返回到请求SM 132。这是因为如果光线未碰撞包围盒(如图3A中关于包围盒310),则光线将不会碰撞被测试的包围盒内的所有其他更小包围盒以及包围盒包含的任何图元。

[0102] 如果由遍历协处理器138执行的包围盒测试显示包围盒与光线相交(步骤514中为“是”),则遍历协处理器确定包围盒是否可以细分为更小的包围盒(步骤518)。在一个示例性实施例中,遍历协处理器138本身不一定执行任何细分。相反,BVH中的每个节点具有一个或更多个子节点(其中每个子节点是BVH中的叶或分支)。对于每个子节点,都有一个包围盒和一个指向分支或叶节点的指针。当光线使用遍历协处理器138处理节点时,它正在针对节点的子节点的包围盒测试自身。光线只将堆栈条目推送到其堆栈上,用于那些其代表性包围盒被碰撞的分支或叶。当光线在示例性实施例中获取节点时,它不测试节点的包围盒-它针对节点的子节点的包围盒进行测试。遍历协处理器138按照由光线配置确定的顺序将其包围盒被光线碰撞的节点推送到光线的遍历堆栈上。例如,可以按照节点在存储器中出现的顺序或者按照它们沿着光线的长度出现的顺序或以某种其他顺序将节点推送到遍历堆栈。如果存在包围盒的进一步细分(步骤518中为“是”),则访问包围盒的那些进一步细分,

并且对每个得到的细分包围盒执行包围盒测试,以确定哪个细分包围盒与光线相交以及哪个不与光线相交。在该递归过程中,可以通过测试514消除一些包围盒,而其他包围盒可以导致通过遍历协处理器138递归地应用步骤512-518来对越来越进一步的细分进行相交测试。

[0103] 一旦遍历协处理器138确定与光线相交的包围盒是叶节点(步骤518中为“否”),遍历协处理器执行图元(例如,三角形)相交测试520,以确定光线是否与相交的包围盒中的图元相交以及光线与哪些图元相交。因此,遍历协处理器138执行相交的后代分支节点的深度优先遍历,直到到达叶节点。遍历协处理器138处理叶节点。如果叶节点是图元范围,则遍历协处理器138针对光线测试它们。如果叶节点是实例节点,则遍历协处理器138应用实例变换。如果叶节点是项目范围,则遍历协处理器138将它们返回到请求SM 132。在示例性非限制性实施例中,SM 132可以命令遍历协处理器138执行不同类型的光线-图元相交测试并取决于来自应用程序(或应用程序在其上运行的软件堆栈)的操作报告不同的结果,并由SM中继到TTU。例如,SM 132可以命令遍历协处理器138报告由相交测试揭示的最近的可见图元,或者报告与光线相交的所有图元,而不管它们是否是最近的可见图元。SM 132可以将这些不同的结果用于不同类型的可视化。一旦遍历协处理器138完成处理叶节点,就可以有其他分支节点(先前推送到光线的堆栈上)进行测试。

[0104] 多个相交

[0105] 更详细地,如图3C所示,任何给定的光线可以与包围盒内的多个图元相交。给定图元内的光线相交是否对可视化产生影响取决于该图元的属性和位置以及SM 132正在执行的可视化过程。例如,图元可以是不透明的、透明的或部分透明的(即半透明的)。不透明的图元将阻挡光线穿过图元,因为眼睛无法透过图元的不透明表面看到。透明图元将允许光线通过(因为眼睛可以透过透明图元看到),但情况可能更复杂。例如,透明图元可能具有镜面特性,这会导致光线的某些部分反射(考虑来自窗玻璃的反射),光线的其余部分通过。其他透明图元用于提供纹理映射到的表面。例如,树的每个个体的叶可以由透明图元建模,叶的图像被纹理映射到该透明图元上。

[0106] 图5A-图5C示出了使用三个三角形的示例的一些场景,这三个三角形被假设在相同的包围盒中并且每个三角形与光线相交。图5A示出了朝向这三个三角形的光线,其中光线相对于不透明的视点遇到第一个三角形。因为“前方”(从眼睛的光线方向看)相交的三角形是不透明的,所以三角形将阻挡光线,因此光线也不会到达其他三角形,即使光线在空间上与它们相交。在该示例中,在识别出不透明三角形的相交之后,可以忽略(剔除)从视点看的不透明三角形“后面”的三角形,因为“前”不透明三角形沿着光线隐藏了用户视图中的其他三角形。在图5A-图5C中用虚线表示剔除。在这种情况下,遍历协处理器138可能仅需要向SM 132报告第一不透明三角形的标识。

[0107] 图5B示出了指向相同三个三角形的光线,但是现在最近的可见三角形是部分透明的而不是不透明的。因为最近的可见相交三角形是至少部分透明的,所以光线可以穿过它以碰撞它后面的不透明三角形。在这种情况下,不透明的三角形将通过部分透明的三角形可见,但会阻挡用户沿着光线的第三三角形的视野。本文,遍历协处理器138可以向SM 132报告两个前面三角形的标识,但是不报告第三个被剔除的三角形,即使光线在空间上与该第三三角形相交。在本文发现顺序可能很重要。在 α 和不透明三角形的情况下,如果首先发

现不透明,则遍历协处理器138将不透明三角形返回到具有遍历状态的SM 132,该遍历状态将在 α 三角形处重新开始测试。虽然本文暗示 α 意味着透明,但它实际上意味着“将我返回到SM 132并让SM决定如何处理它。”例如,可以根据纹理或函数修剪 α 三角形,以便三角形的部分被切掉(即,不存在、不透明)。遍历协处理器138不知道SM 132将如何处理 α 三角形(即,它不处理与修剪的三角形不同的透明三角形)。因此, α 三角形可能阻挡或可能不阻挡或着色沿着光线从点到达它们之外的光,并且在示例性实施例中,它们需要SM 132干预来处理/确定那些事物。

[0108] 图5C示出了光线遇到的前两个三角形是部分透明的场景。因为第一和第二相交三角形是至少部分透明的,所以光线将穿过第一和第二三角形以照射在也相交的第三不透明三角形上。因为第三个相交的三角形是不透明的,它将阻挡光线,并且光线不会照射在第三个三角形后面的任何其他三角形上,即使它们可能在空间上与光线相交。在这种情况下,遍历协处理器138可以将所有三个三角形报告给SM 132,但不需要报告不透明三角形后面的任何其他三角形,因为不透明三角形阻挡光线到达那些附加三角形。

[0109] 然而,在一些模式中,SM 132可能需要知道与光线相交的所有三角形的身份,而不管它们是不透明的还是透明的。在那些模式中,遍历协处理器138可以简单地执行相交测试并返回光线在空间上与之相交的所有三角形的身份(在这种模式中,遍历协处理器将返回图5A-图5C中所示的所有三种情形的相同相交结果),并允许SM 132对其进行分类-或者在某些情况下命令遍历协处理器138对这些相同的三角形进行更多测试。

[0110] 如下面将更详细讨论的,当光线与不透明三角形相交时,遍历协处理器138可以在某些操作中被编程以将被测光线的长度减小到不透明三角形相交的位置,因此它将不报告相交三角形“后面”的任何三角形。当确定部分透明三角形与光线相交时,遍历协处理器138将返回光线照射到的更完整的三角形列表以用于可视化,并且请求SM 132可以执行进一步处理以基于例如三角形的任何纹理或其他属性确定光线是否将被阻挡、传递或部分传递以及部分反射。在示例性实施例中,遍历协处理器138不能访问三角形的纹理属性,因此不会尝试确定关于那些属性的可视化。

[0111] 纹理或其他表面修改

[0112] 例如,图6A和图6B示出了透明三角形610,其具有应用于三角形的叶的纹理615。人们可以想到一个由树脂玻璃制成的三角形,上面有一片叶贴花。如图6A所示,光线620在所施加的纹理615之外的点处与透明三角形610相交。因为光线620与所施加的纹理615外部的三角形相交,所以纹理将不会阻挡光线620并且光线将会不受阻挡地穿过透明三角形610。这就像是能够透过未经叶贴花覆写的树脂玻璃三角形的部分看到。注意,在一个示例性实施例中,SM 132进行可见性确定,因为遍历协处理器138不一定能够访问关于叶贴花的信息。遍历协处理器138通过向SM返回光线与之相交的三角形的识别以及关于该三角形的属性的信息来帮助SM 132。

[0113] 在图6B中,光线630与施加纹理615的透明三角形相交。SM 132将基于纹理615将阻挡光线630还是允许光线630通过来确定遍历协处理器138的后续遍历是否是必要的。如果光线630被纹理615阻挡,则透明三角形610后面的其他三角形(其可能已经与光线630相交)将被纹理阻挡并且不会有助于沿着光线的可视化。在本文的示例性非限制性实施例中,遍历协处理器138不能访问纹理信息,因此它不会尝试加速该确定。遍历协处理器138可以例

如将光线和对象内的各种三角形之间的所有相交返回到请求SM 132,然后SM可以使用图元引擎134进行进一步的光线追踪可视化确定。在其他示例性实施例中,遍历协处理器138可以通过与纹理映射单元和图元引擎134内的3D图形管线的其他部分交互来加速这些测试中的一些或全部,以进行必要的可视化确定。

[0114] 协调变换

[0115] 图2A-图3C仅涉及单个对象,即茶壶。正如你现在所在的房间包含多个对象一样,大多数3D场景都包含许多对象。例如,包含茶壶的3D场景可能还包含杯子、碟子、牛奶罐、勺子、糖碗等,所有这些都放在桌子上。在3D图形中,这些对象中的每一个通常是独立建模的。然后,图形系统100使用来自处理器120的命令将所有模型以期望的位置、方向和尺寸放在公共场景中以便可视化(就像你将设置和安排用于供应茶的桌子一样)。这意味着SM 132可以命令遍历处理器138相对于场景中的多个对象分析相同的光线。然而,当放入公共场景时这些对象中的每一个将在位置、方向和大小上被变换的事实被遍历协处理器138考虑并加速。在非限制性示例性实施例中,从世界到对象空间的变换与世界空间包围盒一起存储在在世界空间BVH中。遍历协处理器138通过将光线从世界(场景)空间变换到对象空间来加速变换过程,以便执行图4所示的测试。特别是,由于几何形状从对象空间到世界(场景)空间的变换是计算密集的,该变换留给图形管线图元引擎134和/或光栅引擎136以作为光栅化的一部分来执行。相反,遍历协处理器138将给定的光线从世界空间变换到由加速数据结构定义的每个对象的坐标系,并在对象空间中执行其测试。

[0116] 图7A和图7B示出了遍历协处理器138如何将相同的光线变换到三个不同的对象空间。图7A示出了桌子上的三个对象:杯子、茶壶和水罐。这三个对象和一个桌子包含一个存在于世界空间中的场景。也在世界空间中定义的光线从视点发出并与三个对象中的每一个相交。

[0117] 图7B示出了在对象空间中定义的三个对象中的每一个。这三个对象中的每一个都由存在于相应对象空间中的相应模型定义。在示例性非限制性实施例中,遍历协处理器138在针对对象执行相交测试之前将光线变换到每个对象的对象空间中。为了遍历协处理器138执行相交测试的目的,该“实例变换”节省了将每个对象的几何形状和加速数据结构的相关盒细分从对象空间变换到世界空间的计算工作量。

[0118] 请求SM 132在一个对象隐藏另一个对象,在另一个对象上投射阴影和/或朝向另一个对象反射光的情况下追踪哪些对象相对于每个个体光线位于哪些其他对象前面并且解析可见性。请求SM 132可以使用遍历处理器138来加速这些测试中的每一个。

[0119] 示例性树形BVH加速数据结构

[0120] 图8A和图8B示出了3D场景(图8A)和相应的树形数据结构(图8B)的递归细分的包围盒,其可以由遍历协处理器138访问并且用于由遍历协处理器执行的硬件加速的操作。包围盒的划分可以用分层树形数据结构表示,其中图2B中所示的大包围盒由树的父节点表示,而较小的包围盒由父节点包含的树的子节点表示。最小的包围盒表示为树中的叶节点,并标识包含在这些最小包围盒内的一个或更多个几何图元。

[0121] 树形数据结构可以存储在遍历协处理器138外部的存储器中,并且基于SM 132向遍历协处理器138发出的查询来检索。树形数据结构包括以层次布置的多个节点。树形结构的根节点N1对应于包围所有三角形01-08的包围盒N1。根节点N1可以识别包围盒N1的顶点

和根节点的子节点。

[0122] 在图8A中,包围盒N1被细分为包围盒N2和N3。图8B的树形结构的子节点N2和N3对应于并表示图8A中所示的包围盒N2和N3。树形数据结构中的子节点N2和N3识别空间中各个包围盒N2和N3的顶点。在该特定示例中,每个包围盒N2和N3被进一步细分。包围盒N2被细分为包含的包围盒N4和N5。包围盒N3被细分为包含的包围盒N6和N7。包围盒N7包括两个包围盒N8和N9。包围盒N8包括三角形07和08,并且包围盒N9包括叶包围盒N10和N11作为其子包围盒。叶包围盒N10包括图元范围(例如,三角形范围)010,并且叶包围盒N11包括项目范围09。图8B的树形结构的各个子节点N4、N5、N6、N8、N10和N11对应于并表示空间中的图8A的包围盒N4、N5、N6、N8、N10和N11。

[0123] 图8B树仅有三到六层深,使得盒N4、N5、N6、N8、N10和N11构成“叶节点”-即,树中没有子节点的节点。图8A示出了叶节点包围盒N4、N5、N6和N8中的每一个都包含场景中的几何形状的两个三角形。例如,盒细分N4包含三角形01和02;盒细分N5包含三角形03和04;盒细分N6包含三角形05和06;以及盒细分N8包含三角形07和08。图8B中所示的树形结构通过将它们与场景几何形状中的适当的三角形01-08相关联来表示这些叶节点N4、N5、N6和N7。为了访问该场景几何形状,遍历协处理器138向下遍历图8B的树形数据结构直到叶节点。通常,树的不同部分可以并且将具有不同的深度并且包含不同数量的三角形。与不包含几何形状的盒细分相关联的叶节点不需要在树形数据结构中明确地表示(即,树被“修剪”)。

[0124] 根据一些实施例,以N7为根的子树可以表示在与对应于节点N1-N3的包围盒不同的坐标空间中定义的一组包围盒或BVH。当包围盒N7与其父包围盒N3处于不同的坐标空间中时,提供遍历以N7为根的子树所必需的光线变换的实例节点N7'可以将树的其余部分连接到以N7为根的子树。实例节点N7'通过定义从N1-N3的坐标空间(例如,世界空间)到N7等的坐标空间(例如,对象空间)的变换,将对应于节点N1-N3的包围盒或BVH与对应于节点N7等的包围盒或BVH连接。

[0125] 遍历协处理器138的内部结构和操作

[0126] 图9示出了遍历协处理器138的示例性简化框图,其包括被配置为执行上述加速遍历操作的硬件(下面描述该遍历协处理器138的更详细的实现)。因为图9中所示的遍历协处理器138适于遍历诸如图8A、图8B所示的基于树的加速数据结构,所以它也可以被称为“树遍历单元”或“TTU”700(附图标记700用于指代图1中所示的遍历协处理器138的更详细的非限制性实现)。树遍历操作可以包括,例如,确定光线是否与包围盒和/或树形数据结构(例如,BVH树)的图元相交,这些测试可以涉及将光线变换到对象空间。

[0127] TTU 700包括用于确定光线是否与包围盒相交的专用硬件和用于确定光线是否与树形数据结构的图元相交的专用硬件。在一些实施例中,TTU 700可以使用采用所支持的叶节点图元的相交测试的短堆栈遍历以及 α 图元和不支持的叶节点图元(项目)的中间遍历返回来执行包围盒层次的深度优先遍历。将参考三角形讨论图元的相交,但也可以使用其他几何图元。

[0128] 更详细地,TTU 700包括相交管理块722、光线管理块730和堆栈管理块740。这些块中的每一个(以及图9中的所有其他块)可以构成由逻辑门、寄存器、硬件嵌入式查找表或其他组合逻辑等实现的专用硬件。

[0129] 光线管理块730负责管理关于由SM 132指定的到光线管理块的光线的信息并执行

关于其的操作。堆栈管理块740与遍历逻辑712结合作以管理关于遍历BVH加速数据结构的信息并执行与其相关的操作。遍历逻辑712由光线-complet测试块710的结果指示,该光线-complet测试块710根据需要使用实例变换测试由光线管理块730指示的光线与由BVH表示的盒细分之间的相交。光线-complet测试块710经由作为TTU 700的一部分的L0完整缓存752从存储器140中检索关于BVH的附加信息。光线-complet测试块710的结果通知遍历逻辑712关于是否需要进一步递归遍历。当遍历逻辑712从BVH的一个级别遍历到另一个级别时,堆栈管理块740维护堆栈以追踪状态信息,当遍历逻辑遍历到BVH更深处时,堆栈管理块将项目推送到堆栈上,当遍历逻辑在BVH中向上遍历时,堆栈管理块将项目从堆栈中弹出。堆栈管理块740能够在SM请求的任何时间向请求SM 132提供状态信息(例如,中间或最终结果)。

[0130] 相交管理块722根据需要使用实例变换来管理关于光线和图元之间的相交的信息并执行与其相关的操作。光线图元测试块720经由作为TTU 700的一部分的L0图元高速缓存754根据需要从存储器140中检索关于几何形状的信息。光线-图元测试和变换块720执行的相交测试的结果通知相交管理块722。因此,光线-图元测试和变换块720向相交管理块722提供相交结果,相交管理块722向请求SM 132报告几何碰撞和相交。

[0131] 堆栈管理单元740检查遍历状态以确定需要检索什么类型的数据以及哪个数据路径(complet或图元)将消耗它。包围盒的相交在TTU 700的光线-complet测试路径中确定,包括一个或更多个光线-complet测试块710和一个或更多个遍历逻辑块712。complet指定包围盒的根节点或内部节点。因此,complet可以为光线-complet测试定义一个或更多个包围盒。TTU 700的光线-complet测试路径识别哪些包围盒与光线相交。需要进一步处理与光线相交的包围盒,以确定与相交的包围盒相关联的图元是否相交。在包括一个或更多个光线-图元测试和变换块720以及一个或更多个相交管理块722的光线-图元测试路径中确定图元的相交。

[0132] TTU 700从一个或更多个SM 132接收查询以执行树遍历操作。查询可以请求光线是否与包围盒和/或BVH数据结构中的图元相交。查询可以识别光线(例如,光线的原点、方向和长度)以及BVH数据结构和遍历状态(例如,短堆栈),其包括引用光线去访问的一个或更多个层次包围盒中的节点的一个或更多个条目。查询还可以包括关于光线在遍历期间如何处理特定类型的相交的信息。光线信息可以存储在光线管理块730中。可以基于光线-图元测试的结果更新存储的光线信息(例如,光线长度)。

[0133] TTU 700可以请求要从TTU 700外部的存储器中检索的在查询中识别的BVH数据结构。BVH数据结构的检索部分可以被高速缓存在TTU 700内的零级(L0)高速缓存750中,因此,该信息可用于其他时间相干的TTU操作,从而减少了存储器140的访问。光线-complet测试所需的BVH数据结构的的部分可以存储在L0complet高速缓存752中,并且光线-图元测试所需的BVH数据结构的的部分可以存储在L0图元高速缓存754中。

[0134] 在所请求的遍历步骤所需的complet信息在complet高速缓存752中可用之后,光线-complet测试块710确定与光线相交的包围盒。在执行该测试时,可以将光线从层次包围盒的坐标空间变换为相对于complet定义的坐标空间。针对与complet的子节点关联的包围盒测试光线。在示例性非限制性实施例中,不针对complet自己的包围盒测试光线,因为(1) TTU 700在测试引用该complet的父包围盒子项时,先前针对类似的包围盒测试了光线,并

且 (2) complet 包围盒的目的是定义一个局部坐标系,其中子包围盒可以用压缩形式表示。如果光线与任何子包围盒相交,则结果被推送到遍历逻辑以确定相应的子指针将被推送到遍历堆栈的顺序(进一步测试可能需要遍历逻辑712向下遍历到BVH的下一级别)。递归地重复这些步骤,直到遇到BVH的相交叶节点。

[0135] 光线-complet测试块710可以向遍历逻辑612提供光线-complet相交。使用光线-complet测试的结果,遍历逻辑712创建要被推送到堆栈管理块740的堆栈条目。堆栈条目可以指示需要通过光线-complet测试块710进一步测试光线相交的内部节点(即,包括一个或更多个子节点的节点)和/或需要通过光线-图元测试和变换块720测试光线相交的相交叶节点中识别的三角形。光线-complet测试块710可以重复在堆栈中识别的内部节点上的遍历,以确定与光线相交的BVH中的所有叶节点。在示例性非限制性实施例中,光线-complet测试块710执行的精确测试将由模式位、光线操作(见下文)和剔除碰撞来确定,并且TTU 700可以将中间结果和最终结果返回到SM 132。

[0136] 相交的叶节点识别可能或可能不与光线相交的图元。一种选择是TTU 700向SM 132提供例如在相交的叶节点中识别的一系列几何形状以进行进一步处理。例如,SM 132自身可以基于TTU 700提供的信息确定所识别的图元是否与光线相交,作为TTU遍历BVH的结果。为了从SM 132卸载该处理并由此使用TTU 700的硬件加速它,堆栈管理块740可以发出对光线-图元和变换块720的请求以对TTU的光线-complet测试块710识别的相交叶节点内的图元执行光线-图元测试。在一些实施例中,SM 132可以发出对光线-图元测试的请求以测试特定范围的图元和变换块720,而不管如何识别该几何形状范围。

[0137] 在确保所请求的光线-图元测试所需的图元数据在图元高速缓存754中可用之后,光线-图元和变换块710可以使用存储在光线管理块730中的光线信息来确定与光线相交的图元。光线-图元测试块720将确定为与光线相交的图元的标识提供给相交管理块722。

[0138] 相交管理块722可以将光线-图元测试的结果返回给SM 132。光线-图元测试的结果可以包括相交图元的标识符,交点与光线原点的距离以及关于相交图元的属性的其他信息。在一些实施例中,相交管理块722可以基于来自光线-图元和变换块710的先前相交结果来修改现有的光线-图元测试(例如,通过修改光线的长度)。

[0139] 相交管理块722还可以追踪不同类型的图元。例如,不同类型的三角形包括在相交时阻挡光线的不透明三角形和在相交时可能阻挡或可能不阻挡光线或者可能需要SM进行额外处理的 α 三角形。光线是否被透明三角形阻挡可以例如取决于映射到三角形上的一个或更多个纹理、纹理占据的三角形区域(参见图6A和图6B)以及纹理修改三角形的方式。例如,在一些实施例中,透明度(例如,彩色玻璃)需要SM 132追踪透明对象碰撞,因此它们可以以光线参数顺序进行分类和着色,并且通常实际上不阻挡光线。同时, α “修剪”允许基于映射到图元上的纹理的形状来修剪图元的形状-例如,从三角形裁剪出叶形(请注意,在光栅图形中,透明度通常被称为“ α 混合”,并且修剪被称为“ α 测试”)。在其他实施例中,TTU 700可以将透明碰撞推送到存储器中的队列,以稍后由SM 132处理,并通过向纹理单元发送请求来直接处理经修剪的三角形。每个三角形可以包括指示三角形类型的指示符。相交管理块722被配置为维护用于追踪不同类型的相交三角形的结果队列。例如,结果队列可以将一个或更多个相交的不透明三角形标识符存储在一个队列中,并将一个或更多个透明三角形标识符存储在另一个队列中。

[0140] 对于不透明三角形,可以在TTU 700中完全确定光线相交,因为不透明三角形的区域阻挡光线穿过三角形的表面。对于透明三角形,在一些实施例中,在TTU 700中不能完全确定光线交点,因为TTU 700基于三角形的几何形状执行相交测试,并且可能无法访问三角形的纹理和/或由纹理占据的三角形区域(在其他实施例中,可以通过图形管线的纹理映射块向TTU提供纹理信息)。为了完全确定三角形是否相交,可以将关于光线-图元和变换块710确定的透明三角形相交的信息发送到SM 132,以便SM完全确定三角形是否影响沿着光线的可见性。

[0141] SM 132可以解决光线是否与关联于透明三角形的纹理相交和/或光线是否将被纹理阻挡。在一些情况下,SM 132可以基于该确定将修改的查询发送到TTU 700(例如,如果光线被纹理阻挡则缩短光线)。

[0142] 在一个实施例中,TTU 700可以被配置为将确定为与光线相交的所有三角形返回到SM 132以进行进一步处理。因为在接口和线程同步方面将每个三角形交点返回到SM 132以进行进一步处理是昂贵的,所以TTU 700可以被配置为隐藏三角形,该三角形是相交的但可证明能够隐藏而不会对结果场景产生功能影响。例如,因为TTU 700具有三角形类型信息(例如,三角形是不透明的还是透明的),所以TTU 700可以使用三角形类型信息来确定沿着光线被另一个相交的不透明三角形遮挡的相交三角形,因此它们不需要包括在结果中,因为它们不会影响沿光线的可见性。如上面参考图5A-图5C所讨论的,如果TTU 700知道一三角形沿着光线被一不透明三角形遮挡,则可以从结果中隐藏被遮挡的三角形而不影响所得场景的可视化。

[0143] 相交管理块722可以包括结果队列,用于存储将三角形ID和关于光线碰撞三角形的点的信息相关联的碰撞。当确定光线与不透明三角形相交时,三角形的身份和交点到光线原点的距离可以存储在结果队列中。如果确定光线与另一个不透明三角形相交,则如果交点到光线原点的距离大于已存储在结果队列中的相交的不透明三角形的距离,则可以从结果中省略另一个相交的不透明三角形。如果交点到光线原点的距离小于已存储在结果队列中的相交的不透明三角形的距离,则另一个相交的不透明三角形可以替换存储在结果队列中的不透明三角形。在测试了查询的所有三角形之后,可以将存储在结果队列中的不透明三角形信息和交点信息发送到SM 132。

[0144] 在一些实施例中,一旦识别出不透明三角形交点,相交管理块722可以缩短存储在光线管理块730中的光线,使得在相交的不透明三角形(沿着光线)后面的包围盒(可以包括三角形)不会被识别为与光线相交。

[0145] 相交管理块722可以将关于相交的透明三角形的信息存储在单独的队列中。存储的关于相交的透明三角形的信息可以被发送到SM 132以供SM解析光线是否与关联于三角形的纹理相交和/或纹理是否阻挡光线。SM可以基于该确定将该确定的结果返回到TTU 700和/或修改查询(例如,如果光线被纹理阻挡则缩短光线)。

[0146] 示例性光线追踪着色管线

[0147] 图10A示出了示例性光线追踪着色管线900,其可由SM 132执行并由TTU 700加速。光线追踪着色管线900由SM 132调用光线生成910并向TTU 700发出相应的光线追踪请求开始。光线追踪请求识别投射到场景中的单个光线,并要求TTU 700搜索具有SM 132也指定的加速数据结构的交点。TTU 700遍历(图10A的框920)加速数据结构,以确定光线与盒细分和

加速数据结构所代表的关联三角形之间的交点或潜在交点。可以通过在加速数据结构中找到与光线相交的包围盒来识别潜在的交点。不需要检查非相交包围盒的后代。

[0148] 对于相交的包围盒内的三角形,TTU 700光线-图元测试块720执行相交930过程以确定光线是否与图元相交。TTU 700将交点信息返回到SM 132,其可以响应于交点确定执行“任何碰撞”着色操作940。例如,SM 132可以执行(或使其他硬件执行)针对相交图元的纹理查找,并且基于适当的纹理元素值来决定如何着色使光线可视化的像素。SM 132追踪这样的结果,因为TTU 700可以以任意顺序返回与场景中不同几何形状的多个交点。

[0149] 或者,可以进一步处理TTU 700确定相交的图元以确定950它们是否应该被着色为未碰撞960或最近的碰撞970。SM 132可以例如指示TTU 700报告指定的几何形状中的最近的碰撞,或者它可以指示TTU报告指定几何形状中的所有碰撞。例如,可以由SM 132实现针对TTU 700基于所实现的环境查找确定相交的图元的“未碰撞”着色操作(例如,通过预先计算的纹理图像的平均来近似反射表面的外观),如图6A和图6B所示。SM 132可以响应于TTU 700提供的针对特定对象几何形状的最接近的碰撞报告,基于材料评估和纹理查找来执行最接近的碰撞着色操作以确定最接近的相交图元。

[0150] 图10B更详细的光线追踪管线流程图示出了用于代表性用例的组件之间的数据流和交互:追踪包含几何图元的场景的光线,其中实例变换在硬件中处理。在一个示例性非限制性实施例中,图10B的光线追踪管线基本上是软件定义的(在示例性实施例中意味着它由SM 132确定)但是通过TTU 700广泛使用硬件加速。关键部件包括SM 132(和计算管线的其余部分)、TTU 700(用作SM的协处理器)以及L1高速缓存和下游存储器系统,TTU从中获取BVH和三角形数据。

[0151] 图10B中所示的管线示出了可以由开发系统提前执行层次包围盒创建1002。它还示出了光线创建和分发1004由示例性实施例中的SM 132或其他软件执行或控制,如着色(其可包括照明和纹理)。示例性管线包括“顶层(top level)”BVH树遍历1006、光线变换1014、“底层(bottom level)”BVH树遍历1018以及每个由TTU 700执行的光线/三角形(或其他图元)相交1026。不必按所示顺序执行,因为TTU 700和SM 132之间的握手(handshake)确定TTU 700做什么以及以什么顺序进行。

[0152] SM 132一次向TTU 700呈现一条或更多条光线。SM 132呈现给TTU 700以进行遍历的每条光线可包括光线的几何参数、遍历状态以及光线的光线标记、模式标记和光线操作信息。在示例性实施例中,光线操作(RayOp)提供或包括辅助算术和/或逻辑测试以抑制、覆盖和/或允许存储交点。SM 132还可以使用遍历堆栈来将某些状态信息传送到TTU 700以用于遍历。可以使用显式遍历堆栈启动新的光线查询。然而,对于某些查询,可以提供少量的堆栈初始化器来开始给定类型的新查询,例如:从complet开始的遍历;光线与一系列三角形的相交;光线与一系列三角形的相交,然后从complet开始的遍历;从三角形缓冲区中获取给定三角形的顶点。在一些实施例中,使用堆栈初始化器而不是显式堆栈初始化提高性能,因为堆栈初始化器需要更少的流式处理器寄存器并减少需要从流式处理器传输到TTU的参数数量。

[0153] 在示例性实施例中,SM 132与每个查询(例如,光线)一起呈现的一组模式标记可以至少部分地控制当查询与特定类型的包围盒相交或与特定图元类型的图元相交时TTU 700将如何处理查询。SM 132提供给TTU 700的模式标记使SM和/或应用程序能够例如通过

RayOp指定辅助算术或逻辑测试来抑制、覆写或允许存储交点。模式标记可以例如使遍历行为能够根据诸如与每个包围盒和/或图元相关联的深度(或距离)、与到原点或光线的距离相关的包围盒或图元的大小等方面而改变。应用程序可以使用此功能来动态地和/或选择性地启用/禁用相交测试的对象集与特定查询集或查询组,例如,以允许在应用程序状态改变时(例如,当门打开或关闭时)使用不同版本的模型,或者提供作为光线长度的函数选择的模型的不同版本以实现细节几何形状级别的形式,或允许来自某些光线类别的特定对象集使某些图层在特定视图中可见或不可见。

[0154] 除了可以针对光线-complet交点和针对光线-图元交点单独指定模式标记集之外,光线数据结构可以指定其他与RayOp测试相关的参数,例如光线标记、光线参数和RayOp测试。TTU 700可以使用光线标记来控制遍历行为、背面剔除和各种子节点类型的处理的各个方面,这取决于可选的RayOp测试的通过/失败状态。RayOp测试增加了TTU 700容量的灵活性,但代价是复杂性。TTU 700为其正在处理的每个活动光线保留“光线槽”,并且可以在遍历期间将光线标记、模式标记和/或RayOp信息存储在TTU内的相应光线槽缓冲器中。

[0155] 在图10B所示的示例中,TTU 700执行顶层树遍历1006和底层树遍历1018。在示例性实施例中,BVH的两层遍历使得能够对动态场景变化进行快速光线追踪响应。

[0156] 光线变换1014通过变换光线提供从顶层树遍历1006到底层树遍历1018的适当过渡,其可以在第一坐标空间(例如,世界空间)中的顶层遍历到底层遍历的BVH的不同坐标空间(例如,对象空间)中使用。在先前的文献中描述了使用两层遍历的示例性BVH遍历技术,参见例如Woop,“动态场景的光线追踪硬件架构(A Ray Tracing Hardware Architecture for Dynamic Scenes)”,萨尔大学,2004,但是实施例不限于此。

[0157] 在一些实施例中,顶层遍历(在世界空间中)是在BVH中进行的,其可以响应于实例化对象的位置或方向的变化而动态地重新计算(例如,由SM 132),并且底层遍历在包围盒的BVH中进行,仍保持静态或基本静态。较低级的BVH中几何形状的变化(其不会改变该较低级的BVH的根节点的包围盒)不需要顶层BVH的变化。用于底层树遍历1018(在对象空间中)的BVH中的包围盒可以包含关于场景几何形状的更详细信息,而不是顶层树遍历1006中使用的相应包围盒,从而避免或至少减少修改底层遍历BVH以响应场景变化。这有助于加速动态场景的光线追踪。

[0158] TTU 700的顶层树遍历1006从L1高速缓存1012接收complet,并且向光线变换1014提供用于变换的实例或者向SM 132提供未碰撞/结束输出1013以用于由SM处理(此块也可以基于非叶节点/无碰撞条件递归操作)的最接近碰撞着色器1015。在顶层树遍历1006中,下一个complet获取步骤1008从存储器和/或高速缓存层次中获取要在步骤1010中测试与光线相交的下一个complet,并且在该获取的complet中的包围盒上完成光线-包围盒相交测试。在示例性实施例中,实例节点被认为是叶节点并且在顶层遍历中被处理。示例非限制性TTU能够在顶层BVH中遇到并处理图元范围和项目范围。每条光线的每个遍历步骤开始于堆栈管理单元检查活动光线的当前活动堆栈。在顶层树遍历中,顶层堆栈处于活动状态,因此堆栈管理单元检查顶层堆栈。如果顶层堆栈处于活动状态且为空,则将光线及其结果队列返回至SM。否则,堆栈管理单元获取该光线要处理的下一个complet或叶的地址,并通过L0/L1高速缓存从存储器子系统请求该complet/叶。如果底层树遍历,底层堆栈处于活动状态,因此堆栈管理单元检查底层堆栈。如果底层堆栈处于活动状态且为空,则光线将切换到

顶层遍历,顶层堆栈将变为活动堆栈。因此遍历可以在底层继续(当底层堆栈变为空时,顶层遍历延续发生)。

[0159] 如上所述,实例节点将一个BVH连接到处于不同坐标系中的另一个BVH。当相交的包围盒的子项是实例节点时,光线变换1014能够从L1高速缓存1016检索适当的变换矩阵。TTU 700使用适当的变换矩阵将光线变换为子BVH的坐标系。已经通过引用并入的美国专利申请No.14/697,480描述了将树中的第一组节点连接到第二组节点的变换节点,其中第一和第二组节点在不同的坐标系中。示例性实施例中的实例节点可以类似于美国申请No.14/697,480中的变换节点。在图10C中所示的TTU 700的替代非实例化模式中,TTU不执行“底”层树遍历1018,并且非实例化的树BVH遍历由框1008、1010执行,例如,仅使用一个堆栈。TTU 700可以基于其从BVH和/或查询类型读取的内容在图10B实例化操作和图10C非实例化操作之间切换。例如,特定查询类型可以限制TTU仅使用非实例化操作。在这样的查询中,任何相交的实例节点都将返回给SM。

[0160] 在一些非限制性实施例中,在获取下一个complet之前,对所获取的complet中的每个包围盒执行步骤1010中的光线-包围盒相交测试。其他实施例可以使用其他技术,例如以深度优先的方式遍历顶层遍历BVH。已通过引用并入的美国专利No.9,582,607描述了可以在示例性实施例中使用的一个或更多个complet结构和内容。美国专利No.9,582,607还描述了complet的示例性遍历。

[0161] 当RCT单元确定光线与子包围盒相交时,树遍历单元记录指向对应子complet或叶节点的指针或对其的引用,以用于随后对光线的测试。在示例性实施例中,一个或更多个堆栈数据结构(例如,图12中所示的遍历堆栈)用于追踪随后要测试与光线的相交的complet或叶节点。在一些示例性实施例中,可以使用小尺寸的遍历堆栈来追踪要由顶层树遍历1006的操作遍历的complet,以及要测试相交的图元,并且可以使用更大的本地堆栈数据结构来追踪底层树遍历1018中的遍历状态。在一些示例性实施例中,用于顶层遍历和底层遍历两者的遍历堆栈可以是短的,即,限于小尺寸(例如,每个有4个条目),以便于在遍历协处理器和SM之间有效地传递堆栈信息。在一些示例性实施例中,顶层堆栈和底层堆栈的大小相等,并且叶节点和complet两者都可以存在于顶层BVH和底层BVH中。在一些示例性实施例中,存在限制在遍历协处理器和SM之间交换的堆栈内容的选项(可以交换的堆栈条目的数量可以独立于堆栈深度来配置)。如果可以在遍历协处理器和SM之间交换的堆栈条目的数量是有限的,则整个堆栈仍然可用于遍历,并且可以限制堆栈的使用方式。例如,使用限制可能是入栈到堆栈上的叶节点可能不超过堆栈限制减去1,除了可以保证在从遍历返回之前从堆栈弹出的所有不透明三角形的情况。

[0162] 在示例性实施例中,当底层BVH遍历是活动的时,上层BVH中的光线的位置和方向与用于遍历底层BVH的经变换的位置和方向分开存储。在底层树遍历1018中,下一个complet获取步骤1022从存储器和/或高速缓存层次1020中获取要在步骤1024中测试光线相交的下一个complet,并在获取的complet中的包围盒上进行光线-包围盒相交测试。如上所述,底层树遍历可以包括其中的包围盒与在上层树遍历中遍历的包围盒处于不同的坐标系中的complet。底层树遍历还接收来自L1高速缓存的complet,并且可以基于无叶/无碰撞条件以及基于未碰撞/结束检测的顶层树遍历1006在其自身内递归地或迭代地操作。可以利用变换到所检索的较低层complet的坐标系的光线来确定光线与较低层BVH中的包围盒

的交点。然后将发现与较低层树遍历中与光线相交的叶包围盒提供给光线/三角形交点1026。

[0163] 如上所述,在示例非限制性实施例中,可以在顶层遍历和底层遍历两者中处理叶节点。例如,底层树遍历1018的叶输出可以被提供给光线/三角形交点1026(其具有L0高速缓存访问以及经由L1高速缓存1028检索三角形的能力)。当到达某些叶节点而不遍历实例的BVH时,光线/三角形交点1026还可以从顶层树遍历1006接收叶输出。L0complet和三角形高速缓存可以是TTU 700内部的小型只读高速缓存。

[0164] 在处理了图元范围中的所有图元之后,相交管理单元检查结果队列的状态并制作分组以发送到堆栈管理单元和/或光线管理单元以更新光线的属性和遍历状态,设置光线的下一个遍历步骤,和/或将光线返回到SM 132(如有必要)。如果结果队列包含在处理图元范围期间发现的不透明或 α 交点,则相交管理单元将结果队列中最近的不透明交点的参数长度(t)发信号到光线管理单元,以记录为光线的上限以缩短光线。要更新遍历状态以设置光线的下一个遍历步骤,相交管理单元向堆栈管理单元发送一下信号:结果队列中是否存在与图元范围的不透明交点,结果队列中是否存在一个或多个 α 交点,结果队列是否已满,是否在图元范围中找到了尚未返回到SM且在结果队列中不存在的额外 α 交点,以及在SM消耗结果队列的内容之后要测试的光线的图元范围中的下一个 α 图元的索引(α 图元之后范围中的下一个图元的索引,具有来自结果队列中当前图元范围的最高存储器顺序)。

[0165] 当堆栈管理单元740从相交管理单元722接收到分组时,堆栈管理单元740检查该分组以确定完成遍历步骤所需的下一个动作并开始下一个动作。如果来自相交管理单元722的分组指示在图元范围中已找到不透明交点并且光线模式位指示一旦找到任何交点则该光线将结束遍历,堆栈管理单元740将该光线及其结果队列返回到SM,遍历状态指示遍历已完成(完成标记设置和/或空顶层和底层堆栈)。如果来自相交管理单元722的分组指示结果队列中存在不透明交点或 α 交点,并且在图元范围(其尚未被返回到SM)处理期间图元范围中还有光线遇到的剩余 α 交点不存在于结果队列中,则堆栈管理单元740将光线和结果队列返回到SM,其中遍历状态被修改以设置剔除不透明位以防止进一步处理图元范围中的不透明图元并且最高 α 图元交点从图元范围返回到光线结果队列中的SM之后,图元范围开始索引前进到第一个 α 图元。如果来自相交管理单元722的分组指示当光线处理了图元范围时没有找到不透明交点或 α 交点,则堆栈管理单元740将堆栈顶部的条目(对应于完成的图元范围)从活动遍历堆栈中弹出。如果来自堆栈管理单元740的分组指示队列中的不透明交点或者结果队列中存在不透明交点,并且一旦找到任何交点和/或结果队列中存在 α 交点,则光线模式位不指示光线将完成遍历,但是在结果队列中不存在的图元范围中没有找到剩余的 α 交点,其尚未返回到SM,堆栈管理单元740从活动遍历堆栈弹出堆栈条目的顶部(对应于完成的图元范围)并修改结果队列的内容,以指示结果队列中存在的所有交点都来自其处理已完成的图元范围。

[0166] 如果活动堆栈是底部堆栈,并且底部堆栈是空的,则堆栈管理单元740将活动堆栈设置为顶部堆栈。如果顶部堆栈是活动堆栈,并且活动堆栈为空,则堆栈管理单元740将光线及其结果队列返回到SM,其遍历状态指示遍历已完成(完成标记设置和/或空顶层和底层堆栈)。如果活动堆栈包含一个或多个堆栈条目,则堆栈管理单元740检查顶部堆栈条目并开始下一个遍历步骤。具有与光线的交点的图元和/或图元范围的测试以及将结果返回

到SM 132在以下申请中描述:共同未决的美国申请No.16/101,148,标题为“保守的严密光线三角交点(Conservative Watertight Ray Triangle Intersection)”(卷号6610-36(18-SC-0145)),美国申请No.16/101,066,标题为“在没有着色器干预下对交点进行连续层次包围盒遍历的方法(Method for Continued Bounding Volume Hierarchy Traversal on Intersection without Shader Intervention)”(卷号6610-32(18-AU-0127))和美国申请No.16/101,196,标题为“用于处理无序的不透明和 α 光线/图元交点的方法(Method for Handling Out-of-Order Opaque and Alpha Ray/Primitive Intersections)”(卷号6610-37(18-AU-0149)),其全部内容通过引用并入本文。

[0167] 虽然以上公开内容在计算机图形和可视化的特定上下文中表达,但是光线追踪和所公开的遍历协处理器可以用于除图形和可视化之外的各种应用。非限制性示例性包括用于逼真声音合成的声音传播,声呐系统的模拟,光学元件和系统的设计,粒子传输模拟(例如,用于医学物理学或实验高能物理),一般波传播模拟,与LIDAR数据的比较,用于例如机器人或车辆定位等目的,OptiX™过去已经用于其中一些应用领域。

[0168] 在不同情况下并且出于各种原因追踪光线。期望能够利用约束来追踪不同类型的光线以将遍历限制到BVH的特定子集或仅针对图元的某些子集测试与光线的相交。这些情况的示例可以包括影响几何形状的对象状态的改变(例如,大门或门的打开/关闭)或着色(例如,交通灯改变),具有不同几何形状或着色细节水平的相同对象的多个版本被选择为沿路径或累积路径长度的反弹次数的函数,省略 α 图元或项目,或突出显示CAD应用程序的特定“层”中的对象。

[0169] 然而,对象和/或要表现的场景中的改变通常可能需要在运行时修改和/或重新生成BVH或其部分。重新生成或甚至修改对应于场景的BVH可能是耗时的。例如,如上所述,BVH中的每个包围盒可以包含许多其他包围盒和多达数百万的几何图元。因此,当要渲染动态场景变化时,需要用于改善光线追踪系统的性能的技术。

[0170] 在一些应用中,场景中的变化和/或可视化的方面可能不需要在BVH中重新生成场景的所有部分。虽然对于许多类型的场景变化,可能需要重建BVH或其部分,在某些情况下,这些变化使得可以在时间上最小化或避免重建或修改BVH。在一些应用程序中,如果光线可以动态地确定如何在同一BVH中表示的同一对象的不同实例或配置之间进行选择,场景中的变化(例如,关闭打开的门)以及从视图中较早的房间内部看到的结果可能不需要改变或重新生成BVH。一些示例性实施例提供了有效处理场景变化的技术。一些其他技术,例如BVH节点的实例化,有助于在发生场景变化时控制BVH的变化。当实例化对象在不改变形状的情况下改变位置和/或方向时,我们只需要更改实例节点的变换系数和包围盒(以及任何未能绑定实例节点的新包围盒的祖先complete包围盒)。这比变换每个图元并将其插入世界空间BVH要少工作。

[0171] 传统的光线追踪器仅基于光线、加速数据结构和图元的几何属性来执行加速数据结构的遍历。虽然许多光线追踪器提供对 α 图元或透明度的支持,但它们在遍历后着色的上下文中这样做。也就是说,虽然许多光线追踪器向某些类型的对象(例如,树的叶子簇和类似于整个不完全不透明的对象)提供将其建模为透明图元,在遍历BVH期间初始确定透明物体可能与光线相交之后,在该遍历之外进行最终确定。为了之后进一步沿着该光线的路径进行测试,这些系统需要发射一条新光线,该光线始于刚好超出透明图元的点,从BVH层次

结构的根开始新的遍历。每次怀疑透明图元交点时使系统启动新遍历的这种要求可能在光线追踪系统产生其输出的速度方面产生严重的性能损失。示例性实施例的优点之一是诸如 α 测试(例如,测试透明图元是否与光线相交)的测试被视为光线相交测试的扩展,并且因此仅短暂地中断遍历,而不是要求为每个遇到的 α 交点发射新的光线。

[0172] 示例性实施例提供对BVH的基本或默认遍历和光线追踪中的着色的各种增强,并提供用于具有在流式多处理器(例如,SM 132)上运行的可编程能力的遍历协处理器(例如,TTU 700)中检测、增强或覆写BVH的硬件加速遍历的机制。

[0173] 如上所述的TTU 700执行用于光线追踪的加速树遍历。如上所述,TTU 700代表在SM 132中运行的线程执行查询,其中,根据一些示例性实施例,每个线程表示相应的光线。也就是说,在某些示例性实施例中,每个线程确定相应光线与BVH的交点。从SM 132获取一个或多个光线(例如,通常,在一些实施例中,来自诸如线程束的一组线程中的每个线程的相应光线)作为输入,TTU 700加速BVH的遍历以便返回光线-图元相交信息(例如,相交的三角形或其他图元),以在随后由SM 132或GPU的另一组件执行的着色处理中使用。示例非限制性实施例提供执行这种加速遍历的方式:通过应用程序实现更精细的控制(例如,在GPU上运行的软件应用程序中的TTU加速的光线追踪渲染),对其执行光线追踪。每条光线声明它将如何处理不同类型的图元和节点,作为用光线指定的模式位和参数以及在BVH的节点和图元内指定的参数的函数。

[0174] 因此,示例性实施例提供用于应用程序微调TTU加速遍历的装置,用于例如但不限于限制遍历加速数据结构的一部分或允许SM上运行的软件中断遍历以覆写或增强加速数据结构中的特定节点或节点种类。

[0175] 为了促进上述遍历过程的特定于查询的动态和可编程改变,示例性实施例提供了每光线的光线操作集(每个光线操作被称为“RayOp”),相关的光线参数、光线标志和模式标志,基于该标志,可以在每光线和/或每交点的基础上改变TTU遍历的行为。一些示例性实施例扩展了美国专利No.9,582,607(其全部内容通过引用并入本文)中描述的树遍历单元,以包括将对交点执行的一组每光线RayOp测试、相关光线参数、光线标志和模式标志等,以及扩展在美国专利申请No.14/697,480(其全部内容通过引用并入本文)中描述的树形结构,以包括用于RayOp测试的每子节点参数(称为“rval”参数)。RayOp测试是在遍历期间执行的光线相交测试的补充。例如,在一个非限制性实施例中,仅对已经与光线相交的子节点和图元执行RayOp测试。虽然光线相交测试是基于光线和BVH的几何方面(例如,位置坐标、方向、长度、大小等),但至少一些RayOp测试可能不基于光线或BVH的几何方面。

[0176] TTU通过使用光线的RayOp标识符(在本公开中称为光线的RayOp操作码)、光线参数和每个潜在遍历步骤处的子节点的rval参数评估预定的RayOp测试表达来对光线执行RayOp测试。根据一个实施例,RayOp使用两个每查询RayOp测试参数A、B和为节点的每个子节点指定的rval指定要针对在遍历期间测试的每个节点的每个子节点执行的每光线测试。示例性RayOp测试提供基于关于特定算术或逻辑操作的光线参数,相对于基于光线参数和相交节点的参数(例如,rval参数)的右手侧值,测试左手侧数值。如下所述,可以在各种实施例中使用其他形式的RayOp测试。

[0177] 根据RayOp测试的结果和光线模式标志,TTU可以确定它将如何处理在遍历期间遇到的不同节点类型和图元碰撞类型-例如,是否应该对子节点进行遍历、忽略或返回至SM进

行特殊处理,或者是否应该对图元交点进行抑制、保留或返回至SM以进行特殊处理。还可以进行确定以便与一个或更多个光线标志一致。例如,光线的光线标志可以指定包围盒(例如,子包围盒)的遍历顺序,从遍历返回时是否弹出堆栈(例如,表示遍历状态的堆栈),当光线的时间间隔在节点的包围盒内开始时是否向SM报告节点参考,是否在交点的第一次碰撞时返回前向设置和剔除设置。

[0178] 模式标志指定TTU在它应如何处理特定节点类型和图元测试结果中的行为,作为RayOp测试是通过还是失败的函数。指定的模式标志可以被认为是两类:模式标志,其影响响应于相交的子包围盒由光线-complet测试(RCT)单元(例如光线-complet测试块710)传递到堆栈管理单元的遍历状态,以及模式标志,其影响传递到堆栈管理单元的遍历状态,和作为由光线图元测试子单元(例如,光线-三角形测试块720)测试的图元的结果由相交管理单元(例如,块722)推送到结果队列的值。前者可以称为RCT模式标志,后者可以称为RPT模式标志。

[0179] 当RCT单元710与子包围盒相交时,RCT单元使用光线的A,B参数和complet子项的rval参数(以及可选地,反转参数)执行由光线的RayOp查询参数指定的RayOp测试。RCT单元710基于RayOp测试的结果(或者如果设置了子节点的反转标志,则为RayOp测试的结果的反转(inverse))来执行由适当的RCT模式标志枚举的动作。例如,如果RayOp=EQUAL,A=0x0和B=0xFF的光线与rval为0x1并且其反转字段设置为FALSE的complet子节点相交,则RayOp测试(例如,0x0==0x1和0xFF)将失败。基于RayOp测试的结果,标识适当的模式标志。例如,如果标识的模式标志(例如,ch_f模式标志)指示当他们的RayOp测试失败时要剔除complet子节点,则TL 712可能不会将堆栈条目推送到该complet子节点的光线的遍历堆栈上,即使光线与complet子项的包围盒相交。在一些实施例中,RCT单元710执行RayOp测试,并将测试结果传递给TL子单元712,TL子单元712使用测试结果在可用模式之间进行选择。

[0180] 上面描述的模式标志是示例,并且关于图10B和图10C以及图14-图16描述了更多模式标志及其相应的配置。

[0181] 在RayOp和模式标志指示将结果返回到流式处理器以进行特殊处理的情况下,光线、其遍历状态及其结果队列内容可以整体返回到SM.SM可以根据需要或根据指定执行附加处理(例如,内插纹理坐标并评估 α 纹理以确定是否要忽略 α 交点等),并且如果需要,SM可以使用相同或更改的查询参数重新提交查询(例如,使用缩短的光线重新提交查询)到TTU。

[0182] 在一些实施例中,还提供了一种机制,以将在RCT测试块710中执行的RayOp测试的结果传送到RTT测试块720以供后续使用。

[0183] 通过提供可以在遍历协处理器(诸如TTU 700)中定制每个光线的遍历和图元相交测试行为的查询结构和BVH结构,应用程序可以影响光线追踪渲染中的动态变化,至少在某些情况下,无需重新生成、修改或复制BVH数据。改变遍历的默认行为可以包括改变遍历的状态(例如,选择BVH的不同部分用于遍历或选择不同的顺序或从默认配置的行为遍历)或改变返回至SM的返回的输出(例如,改变一个或更多的交点)。

[0184] 以上结合图10B和图10C描述了该机制,但是下面提供了可以包括在BVH内并且在遍历期间遇到的遍历树的不同类型的节点的简要概述,以及TTU 700可以返回到SM的各种碰撞类型。

[0185] 如上所述,BVH包括层次布置(例如,在树形结构中)的一组包围盒,每个包围盒包含场景的三维部分。BVH树的叶节点可以包括几何图元(例如,三角形)范围、项目、项目范围或实例节点。如上所述,图8A示出了递归地细分为包围盒的示例性3D场景,图8B示出了可以表示图8A中的3D场景的示例性树形数据结构。

[0186] 本说明书可互换地使用术语“层次包围盒”(BVH)和“加速结构”。示例性实施例主要描述为使用BVH作为加速结构。然而,本领域技术人员将理解,某些示例性实施例可以包括除了本公开中描述的BVH结构的类型之外的加速结构。

[0187] 在一些示例性实施例中,BVH存储在一个或更多个complet中,所述complet是用于存储树形结构中的BVH的节点集的压缩格式。如上所述,每个complet指定根、内部节点和叶节点,每个节点表示BVH的相应包围盒。每个complet的节点可以存储在连续的存储器中。

[0188] 在遍历操作期间,从存储器中以complet形式检索BVH到TTU中,应用软件将BVH写入到存储器中。为了有效检索,complet的大小可以被配置为对应于硬件的高速缓存和/或其他存储器访问能力。例如,在一些实施例中,每个complet是128B,对应于高速缓存行的大小。因为,如上所述,每个complet包括根节点和子节点,所存储的BVH可以被认为是complet的树。为了有效遍历,父complet的complet子项优选地连续存储在存储器中,其中指向complet子项的指针存储在父complet中。指针可以以压缩形式存储。

[0189] BVH的complet本身被布置在树形结构中,complet的示例性子节点类型包括complet、实例节点、项目或项目范围和图元或图元范围。

[0190] 当在遍历期间遇到complet子项时,根据一些配置的默认行为是TTU 700通过从存储器检索complet子项并且下降到complet子项以遍历complet子项的节点来继续遍历。默认行为也可以根据complet子项的节点类型而有所不同。

[0191] complet的实例子节点可以包括指向BVH的节点的层次结构的一个或更多个指针,或者更具体地指向与实例节点和包括该实例节点作为子节点的complet不同的坐标空间中的另一个complet的指针,以及用于将光线从世界空间或其他坐标空间变换到引用的complet的坐标空间的变换。BVH可以包括零个或多个实例节点,每个节点提供来自一个complet的叶节点到另一个complet的根节点的连接。当BVH被认为包括两个或更多个较小的BVH时,可以认为实例节点提供从第一坐标系中的第一BVH到第二BVH的连接,第二BVH被布置为第二坐标系中的第一BVH的子项。根据一些实施例,除了指向引用的complet或BVH的一个或更多个指针之外,实例节点结构可以包含可用于影响引用的complet或者BVH中的行为(例如,背面剔除行为)的标志,以及从引用complet或BVH的坐标系到引用的complet或BVH的坐标系的变换矩阵(在齐次坐标中)。变换可以是世界空间到被引用的complet或BVH的对象空间,或者从引用的complet或BVH的第一对象空间到被引用的complet或BVH的第二对象空间。引用complet或BVH的对象空间的局部坐标系可以被定义为应用于引用complet或BVH的空间(例如,世界空间或另一个对象空间)的坐标系的平移。根据一些实施例,除了将引用的complet或BVH的全局或其他局部坐标系的原点变换为局部坐标系的新原点之外,可以将旋转和/或缩放变换应用于全局或其他局部坐标系以定义引用的complet或BVH的局部坐标系。

[0192] complet的项目范围子项指定一个类型的一个或更多个图元的组,其未被TTU测试相交。根据一些实施例,TTU将项目范围的相交测试委托给在SM上运行的软件。例如,当在遍

历期间遇到由索引(例如,可以从存储在complet中的用户或软件提供的“项目范围基础”导出)组成的项目范围和项目的例如6位计数时,在默认情况下,项目范围作为“项目范围(ItemRange)”碰撞类型返回到SM 132,包括例如索引(例如,47位索引),计数(例如,6位计数)和与叶包围盒的交点的t值(例如,沿着光线的坐标)。

[0193] complet的图元范围(在一个实施例中,三角形范围)指定一组一个或更多个图元类型的压缩或未压缩的 α (例如,透明)或不透明图元,TTU可以对其执行相交测试,用于由存储在连续块阵列中的TTU处理的一些或所有类型的查询。零或更多图元缓冲器包含存储为例如三元组或其他组的例如每顶点的fp32坐标或参数或者由TTU 700可读的无损压缩格式的图元。图元范围参数可以包括非顶点参数(例如,半径)。另外,可以为每个图元提供 α 位,指示用户或软件已经确定需要由软件进行特殊处理以确定图元是否实际上与特定光线相交的图元。在某些示例性实施例中,图元缓冲器可以被组织成存储器和/或高速缓存对齐的块,例如,128B大小和128B对齐。在一个实施例中,三角形范围指定一个或更多个压缩或未压缩的 α (例如透明)或不透明三角形的组,TTU可以测试这些三角形与光线的交点。

[0194] 在一个实施例中,当在遍历期间遇到对应于三角形缓冲器内的连续三角形集合的三角形范围时,默认情况下,在某些示例性实施例中,由TTU 700通过测试范围内三角形的相交并相应地缩短光线来本地处理光线遇到的三角形范围。如果遍历完成并且三角形是相交的,则默认行为是将三角形ID以及交点的t值和重心坐标返回到SM 132。这可以称为“三角形”碰撞类型。在一些实施例中,默认情况下,即使遍历尚未完成,也将具有 α 位设置的相交三角形返回到SM 132。如下面更详细描述,如果软件确定三角形实际上是透明的,则返回到SM的具有交点的遍历堆栈可以包含继续遍历所需的状态。在一些实施例中,针对光束不支持三角形交点,因此默认情况下,所遇到的三角形范围可以作为“TriRange”碰撞类型返回到SM 132,其包括指向与范围重叠的第一三角形块的指针,指定范围的参数和交点与叶包围盒的t值(例如,交点的参数距离)。

[0195] 呈现用于由流式处理器(例如,SM 132)查询树遍历单元(例如,TTU 700)的每条光线包括:(1)光线几何参数,例如光线的位置、方向矢量,以及近和远参数区间(t_{min} , t_{max});(2)遍历状态,其可以包括一个或更多个堆栈条目的堆栈,该堆栈条目引用BVH树形结构中的块,这些块将被提取并针对光线测试交点;(3)模式标志,描述光线在遍历和RayOp信息期间如何处理特定类型的交互。

[0196] 对于每条光线,可以提供光线的几何信息,例如光线的原点、方向以及近和远参数区间。原点、方向以及参数区间的起点和终点可以各自指定为坐标元组。近和远参数位置在本文中分别称为 t_{min} 和 t_{max} ,并且可以被认为表示沿着光线的当前感兴趣的区段。在一些实施例中,可以在查询中指定波束。另外由第二原点和方向指定光束。

[0197] 图11A中示出了示例查询或光线数据结构1102。在一些示例性实施例中,通过SM中的寄存器将光线信息传送到TTU。在其中光线数据经由存储器传递到TTU的示例性实施例中,数据结构1102或其一部分可以驻留在存储器中以由TTU读取。光线数据结构1102可以包括光线标识符1104(其可以由TTU或SM分配以唯一地标识在TTU中同时处理的光线)、光线原点1106、光线方向1108、光线起点(t_{min})1110和终点(t_{max})1112参数。根据一些实施例,光线信息1102还可以包括光线标志1114、RCT模式标志1116(也称为RCT模式位)、RPT模式标志1118(也称为RPT模式位)和一个或更多个光线操作(RayOps)规范。每个RayOps规范可以包

括光线操作码1120和光线测试参数(例如,光线参数A和B)1121。这些光线数据属性在下面描述。

[0198] 如下所述,使用光线的RayOp操作码、模式位、参数A和B以及由每个相交的complete子项或图元指定的一个或多个参数(例如,ChildType、“rval”参数或“ α ”标志)执行针对与光线相交的每个图元或子包围盒的“RayOp”测试。在示例性实施例中,在下面描述的RayOp测试中使用的ChildType和rval参数是针对complet中的每个子项或者针对作为整体的complet指定的,并且为每条光线指定RayOp操作码、模式位和参数A和B。数据结构1122的示例可以保持与RayOp相关的信息,或者根据一些实施例的complet或包围盒在图11B中示出。根据一些实施例,数据结构1122可以通过软件存储在存储器中,并且TTU可以访问存储器中的数据结构和/或可以将数据结构接收到TTU内部存储器中。数据结构1122可以包括报头信息1124,一个或多个覆写标志1126和rval 1128。报头信息可以包括与节点有关的几何信息、节点类型信息等。下面描述覆写标志1226和rval参数1128。

[0199] 在TTU中由光线遍历BVH期间,光线的遍历状态保持在TTU中。遍历状态可以包括一个或多个条目的堆栈,其引用树形结构中的包围盒和/或complet,包围盒和/或complet将被提取并针对光线进行测试。根据一些实施例的遍历堆栈1202在图12中示出。遍历堆栈1202可以包括任何数量的堆栈条目。在一些实施例中,堆栈1202限于少量条目(例如,4个条目的“短堆栈”),使得可以使TTU和SM之间的堆栈交换更有效。在图12中,示出了底部堆栈条目1204和顶部堆栈条目1206,其之间具有一个或多个条目。

[0200] 在至少一些实施例中,TTU 700是无状态的。也就是说,在查询之间在TTU中没有维护架构状态。遍历堆栈使TTU能够在查询期间保持状态,并根据需要将该状态传递给SM。在SM上运行的软件可以通过将包含在遍历堆栈中的状态传递回TTU来请求继续先前的查询。SM使用一个或多个堆栈或具有特定大小堆栈限制(SL)的其他数据结构向TTU传送每个查询的遍历状态,并且TTU使用一个或多个堆栈或其他大小SL的数据结构将遍历状态返回到SM。在TTU内,每条光线保持一个或多个堆栈或其他数据结构,其大小可以 $\geq$ SL。

[0201] 图13示出了根据一些实施例的示例性结果队列。如在别处所述,结果队列用于TTU以将关于到目前为止检测到的交点的信息发送到SM。在一些实施例中,结果队列1310很小并且可以仅容纳不透明图元交点结果1312和/或一个或多个 α 图元交点结果1314。然而,在其他实施例中,结果队列可以容纳表示检测到的交点的更多条目。

[0202] 上面图10B和图10C的图表总结了光线追踪管线并显示了代表性用例的组件之间的数据流和交互:追踪包含图元(诸如三角形)的场景中的光线,实例变换和某些相交测试在硬件中处理。如上参照图10B所示,可以一次向TTU呈现一条或更多条光线以供SM查询,并且呈现给TTU以进行遍历的每条光线可以包括光线的几何参数、遍历状态以及光线的标志、模式标志和RayOp信息。图10B和图10C的遍历操作可以使用诸如图11A中所示的光线数据结构,诸如图11B中所示的节点数据结构,诸如图12中所示的一个或多个遍历堆栈,以及诸如图13中所示的结果队列。在一些实施例中,结果队列可以是保持不透明或 α 结果的单个条目。

[0203] 模式标志使得遍历行为能够通过例如指定辅助算术或RayOp形式的逻辑测试来改变,以抑制、覆写或允许存储在诸如图10B所示的遍历期间遇到的交点。如图10B所示,应用程序可以使用此能力来动态地和/或选择性地启用/禁用用于交点测试的对象集与特定查

询集或查询组,例如,以允许在游戏状态改变时使用不同版本的模型(例如,当门打开或关闭时)或提供不同版本的模型,这些版本被选择作为光线长度的函数来实现几何级别的细节形式,或者允许来自某些类别的光线的特定对象集使某些图层在特定视图中可见或不可见。此功能还允许将 α 更改为不透明,反之亦然。模式标志可以使得遍历行为能够根据诸如与每个包围盒和/或图元相关联的深度(或距离),包围盒或图元的大小相对于原点或光线、物体的特定实例等的距离来改变遍历行为。

[0204] 与每个查询(例如,光线)一起呈现的模式标志集可以至少部分地控制当查询与特定类型的包围盒相交或与特定图元类型的图元相交时TTU将如何处理查询。除了可以为光线-complet交点和光线-图元交点单独指定模式标志集之外,光线数据结构还可以指定其他与RayOp测试相关的参数,例如光线标志、光线参数和RayOp测试。TTU为在TTU中处理的每个活动光线保留“光线槽”,并且可以在遍历期间将光线标志、模式标志和/或RayOp信息存储在TTU内的相应光线槽缓冲器中。

[0205] TTU可以使用光线标志来控制遍历行为的各个方面、背面剔除和各种子节点类型的处理,服从可选的RayOp测试的通过/失败状态。因此,RayOp测试增加了TTU能力的灵活性,但代价是复杂性。

[0206] 图14示出了当在光线追踪管线处理期间检测到光线-包围盒相交时可以执行的过程1400的流程图。例如,相对于图10B中所示的过程,当在图10B中所示的步骤1010和/或1024中检测到光线-包围盒相交时(例如,在顶层遍历和/或底层遍历中),可以执行过程1400。可以在光线-complet测试块710中的TTU 700中执行光线-包围盒相交测试1010和/或1024。

[0207] 步骤1402处的交点检测可以在测试检索到的complet时发生,或者更具体地,在测试检索到的complet中包括的子包围盒时发生。根据示例性实施例,当处理complet时,TTU可以可选地对每个子项执行RayOp测试。在一些实施例中,仅对其相应的包围盒与光线相交的子项运行RayOp测试。

[0208] 因此,在步骤1404,确定所获取的complet具有至少一个子项,并且在步骤1406,并行访问和测试子包围盒。在一些实施例中,每个检索的complet具有零个或一个父complet和零个或更多个complet子项以及零个或更多个叶节点子项。在一些实施例中,每个获取的complet使用父指针或偏移引用其父complet,以压缩形式编码子指针,并提供包含子包围盒和RayOp测试使用的每子数据的每子结构(例如Rval1,反转RayOp结果标志)和(在叶节点的情况下)用于寻址和处理叶节点块的数据(例如,项目计数、起始图元索引、叶中的块数、指示存在 α 图元的标志)。在一些实施例中,可以针对所有子包围盒并行地执行处理步骤1408-1414。在其他实施例中,处理步骤1408-1414可以针对子包围盒的组等并行地逐个子项执行。

[0209] 相交的父包围盒的每个子包围盒是潜在的遍历目标。在示例性实施例中,实例节点是指向另一个BVH的根节点的叶节点。在确定是否检索对应于用于遍历的相应子节点的complet之前,可以基于已经检索的complet中可用的子包围盒信息对相交的父节点子节点执行RayOp测试。

[0210] 在步骤1408,针对所访问的子包围盒执行针对光线指定的RayOp测试。如上面关于图11A所述,可以将RayOp操作码指定为从SM 132提供给TTU 700的光线数据的一部分。在示

例性实施例中,当在光线-complet测试块710处检测到光线-包围盒交点时,遍历逻辑块712可以基于光线和相交的包围盒的子节点执行RayOp测试。更具体地说,由为光线指定的特定RayOp操作码指定的RayOp测试是使用为子包围盒指定的光线的RayOp A、B参数和RayOp rval参数执行的。在一些实施例中,仅针对自身被发现与光线相交的子包围盒执行RayOp测试。例如,当RCT单元针对complet测试光线时,还测试每个complet的子包围盒与光线的交点,并且对于发现与光线相交的每个子包围盒执行RayOp测试。

[0211] 示例性RayOp测试可以提供基于关于特定算术或逻辑运算的光线参数测试左手侧数值,基于光线参数和相交节点参数测试右手侧数值。RayOp测试可以是算术或逻辑计算,其导致真/假输出。特定计算(例如,RayOp A和B参数之间的特定关系,RayOp操作码和rval参数)可以是可配置的,和/或可以在硬件中预编程。在一些实施例中,每条光线可以指定对应于相应的RayOp测试的多个操作码中的一个。因此,RayOp测试提供了高度灵活的技术,通过该技术,光线可以在个体或组的基础上改变TTU 700的默认光线追踪行为。

[0212] RayOp测试可以包括但不限于算术和/或逻辑运算ALWAYS、NEVER、EQUAL、NOTEQUAL、LESS、LEQUAL、GREATER、GEQUAL、TMIN_LESS、TMIN_GEQUAL、TMAX_LESS、TMAX_GEQUAL作为操作码。在一些实施例中,在光线中指定的操作码可以是任何逻辑或算术运算。

[0213] 例如,如果光线的RayOp操作码在提供给TTU的光线信息中定义为“EQUAL”,并且RayOp A和B参数分别是0x0和0xFF,并且所访问的子包围盒的RayOp rval是0x1,RayOp测试可能是“A EQUAL rval&&B”。因此,利用上述各种参数和操作码的值,RayOp测试产生“0x00 == 0x1&&0xFF”。因此,(因为这是假),此示例中的RayOp测试必须返回假。也就是说,在此特定示例中,对于光线和访问的子包围盒的RayOp测试失败。

[0214] 在一些实施例中,子包围盒还可具有与RayOp测试相关联的反转(“eg,inv”)参数。如果光线还具有与RayOp相关联的反转参数,并且反转参数设置为TRUE(例如,1),则返回的RayOp结果可以是实际RayOp测试结果的反转。例如,如果光线的反转参数设置为TRUE,则上例中的RayOp测试将返回TRUE。RayOps可能与光栅图形中的模板测试(Stencil Test)比较,但模板测试能够允许片段写入在片段未通过深度测试时也会发生。在示例性实施例中,RayOps不具有将未碰撞的complet子项转换为碰撞complet子项的能力,但是在其他实施例中,TTU可以允许可编程性,因此RayOp可以将未碰撞视为碰撞。

[0215] RayOp测试不必具有以关系(诸如“A EQUAL rval&&B”)排列的参数和操作码。示例性实施例可以使参数和操作码以任何逻辑或算术关系排列。在一些实施例中,例如,该关系可以是诸如“TMIN_LESS rval”或“TMIN_LESS A&rval”的形式,表示指定的感兴趣区域与单独的节点参数或光线参数的组合之间的关系。示例操作码TMIN_LESS、TMIN_GEQUAL、TMAX_LESS、TMAX_GEQUAL都使RayOp测试能够基于交点的起点或终点(例如,上述操作码中的TMIN和TMAX可以分别表示相交的盒体上光线入口处的t值和相交的盒体上光线出口处的t值(例如,下面的bbox.tmin,bbox.tmax)),并且包括单独测试的节点或测试的节点以及光线参数A和/或B的方面。例如,当rval用于节点的距离值编码时,则“TMIN_LESS rval”可以表示诸如“测试节点的距离小于兴趣区域的开始?”的测试。除了光线的开始/结束之外的基于光线的方面的操作码也是可能的,并且在其他实施例中可以用于RayOp。与编码光线几何属性方面的操作码相比,示例操作码ALWAYS、NEVER、EQUAL、NOTEQUAL、LESS、LEQUAL、GREATER、GEQUAL使得任意指定的左侧值可以与任意指定的右侧值进行比较。因此,示例操作码

ALWAYS、NEVER、EQUAL、NOTEQUAL、LESS、LEQUAL、GREATER、GEQUAL可用于取决于光线或测试节点的某些几何方面的RayOp测试,此外,可以用于独立于光线和测试节点之一或两者的任何几何属性的RayOp测试。因此,在示例非限制性实施例中,“FLT_TMIN_LESS”、“FLT_TMIN_GEQUAL”和“FLT_TMAX_LESS”以及“FLT_TMAX_GEQUAL”RayOp测试实际上分别评估表达式 $\text{bbox.tmin} < A * \text{rval} + B, \text{bbox.tmin} \geq A * \text{rval} + B, \text{bbox.tmax} < A * \text{rval} + B, \text{bbox.tmax} \geq A * \text{rval} + B$ 。在一个特定的非限制性实施方案中,rval是FP0.6.2值,A和B是这些操作的FP1.5.10值。此外,在一些非限制性示例性实施例中,由于FLT_TMIN和FLT_TMAX测试在包围盒tmin和包围盒tmax值(其可以是在相交测试中计算的几何值)上操作,因此这些RayOps可以用于几何细节水平(例如,其中A对应于圆锥对图像平面像素的角的余弦,B对应于光线的先前光线反弹的累积长度,并且rval对应于包围盒的最大长度)。在一些实施例中,操作码(例如,FLT_TMIN_LESS、FLT_TMAX_LESS)用于将在光线/加速数据结构相交测试期间计算的值与关联于至少一个节点的至少一个几何参数进行比较,该测试由与光线相关联的一个几何属性缩放并且由与光线相关联的另一几何属性偏置。

[0216] 在步骤1410,标识对应于RayOp测试结果的一个或多个模式标志。例如,可以在光线数据结构中的预定位位置中指定每个模式标志,并且每个模式标志可以包括任何数量的位。每个模式标志将RayOp测试的结果或者RayOp测试的结果与测试节点的节点类型的组合映射到TTU要采取的特定动作。在一些实施例中,模式标志针对光线-complet测试和光线-图元测试用光线分别指定。因此,响应于在步骤1410完成RayOp测试,可以在为光线指定的RCT模式标志中找到适用的一个或多个模式标志。

[0217] 在以上示例中,由于RayOp测试失败,所以可应用的一个或多个模式标志包括“ch_f模式标志”。如上所述,“ch_f”表示对于类型为complet的相交子项,RayOp测试失败。

[0218] 在步骤1412,标识基于所标识的一个或多个模式标志和/或光线标志要执行的动作,并且在步骤1414执行动作。

[0219] RCT模式标志表示针对每个complet子项类型(例如,complet、实例叶节点、项目范围叶节点、图元范围叶节点)TTU如何处理光线与用于通过或未通过RayOp测试的那些光线的类型的子节点的子包围盒的相交。示例性RCT模式标志包括“In_f”、“In_p”、“Ir_f”、“Ir_p”、“pr_f”、“pr_p”、“ch_f”和“ch_p”。

[0220] 模式标志“In_f”(“modeInstanceNodeFail”)指定当针对类型实例节点(“InstanceNode”)的相交子节点的RayOp测试失败时要执行的动作。支持的动作可以包括在TTU中的处理,剔除(例如,抑制实例节点推送到遍历堆栈上),作为节点参考返回,或返回到SM。

[0221] 模式标志“In_p”(“modeItemRangePass”)指定在针对类型实例节点的相交子节点的RayOp测试通过时要执行的动作。支持的动作可以包括在TTU中的处理,剔除(例如,抑制实例节点推送到遍历堆栈上),作为节点参考返回,或返回到SM。

[0222] 模式标志“Ir_f”(“modeItemRangeFail”)指定在针对类型项目范围(“ItemRange”)的相交子节点的RayOp测试失败时要执行的动作。所支持的动作可以包括返回至SM(例如,将项目范围碰撞推送到结果队列中),剔除(例如,抑制项目范围碰撞存储在结果队列中),或者作为节点参考返回。

[0223] 模式标志“Ir_p”(“modeItemRangePass”)指定在针对类型项目范围的相交子项的

RayOp测试通过时要执行的动作。支持的动作可以包括返回至SM(例如,将项目范围碰撞推送到结果队列中),剔除(例如,抑制项目范围碰撞存储在结果队列中),或者作为节点参考返回。

[0224] 模式标志“pr_f”(“modePrimitiveRangeFail”)指定针对类型图元范围(“PrimitiveRange”)的相交子项的RayOp测试失败时要执行的动作。所支持的动作可以包括在TTU中的处理(例如,将条目推送到遍历堆栈),剔除(例如,抑制将三角形范围堆栈条目推送到遍历堆栈上),作为节点参考返回,或返回到SM。

[0225] 模式标志“pr_p”(“modePrimitiveRangePass”)指定针对类型图元范围的相交子项的RayOp测试通过时要执行的动作。所支持的动作可以包括在TTU中的处理(例如,将条目推送到遍历堆栈),剔除(例如,抑制将图元范围堆栈条目推送到遍历堆栈),作为节点参考返回,或返回到SM。

[0226] 模式标志“ch_f”(“modeCompletFail”)指定针对类型complet(“complet”)的相交子项的RayOp测试失败时要执行的动作。支持的动作可以包括在TTU遍历,剔除或作为节点参考返回。

[0227] 模式标志“ch_p”(“modeCompletPass”)指定当针对类型complet的相交子项通过RayOp测试时要执行的动作。支持的动作可以包括在TTU中遍历,剔除或作为节点参考返回。

[0228] 在一些实施例中,除了根据RayOp测试结果选择的一个或多个模式标志之外,可以以与光线数据中指定的一个或多个光线标志一致的方式执行所选择的动作。光线标志(诸如光线标志1214)可以指定独立于任何特定相交的行为。

[0229] 在示例性实施例中,光线标志可以指定包围盒的遍历顺序,是否在返回时弹出堆栈,是否在光线的tmin...tmax间隔在节点的包围盒内开始时向SM报告节点参考,是否在交点的第一次碰撞时返回,前向设置,剔除设置等。

[0230] 遍历顺序的光线标志可以指定以下中的任何一个:沿着光线的参数距离的遍历,以包围盒和/或图元的存储顺序遍历,减小x坐标,增加x坐标,减小y坐标,增加y坐标,减小z坐标,增加z坐标等。更具体地,遍历顺序指示当complet子包围盒与光线相交时堆栈条目被推送到遍历堆栈上的顺序。特别地,当节点相交时,TTU可以使用由光线标志指定的遍历顺序来确定将相交节点的子节点以哪种顺序推送到遍历堆栈中。例如,对于追踪被设置为在找到第一个碰撞而不是特别是最近的碰撞时返回的阴影光线是有用的,这对于这样的光线首先针对较大的图元进行测试是合乎需要的(因此更可能被碰撞)。如果以这样的方式构建BVH,使得叶节点子节点的存储顺序是最大最先的-那么期望为这样的光线选择超过t排序的存储顺序,因为对于这样的光线来说它更可能更快地返回到SM,t排序并不重要。

[0231] 由于若干原因中的任何一个,可能希望改变遍历顺序(t排序)。例如,当试图找到最接近的三角形时,通常会想要使用-排序,以便首先测试参数长度较早的那些图元。如果这些图元相交,则可能不需要测试沿着光线更远图元和complet。当试图找到任何交点(例如,以测试点是否在光的阴影中)时,人们可能不关心哪些特定图元相交并且可能想要测试最可能首先相交的图元。在这种情况下,BVH构建器可以在树中更早地放置最大的三角形,使得存储器顺序将首先找到它们。

[0232] 在t排序可能不一致的情况下,遍历每个x/y/z排序可用于近似t排序。具体地,光束遍历和光线遍历的t交点可能不一致,因为查询是不同的形状(例如,它们可能相似但不

相同)。然而, $x/y/z$ 排序各自仅基于包围盒位置, 并且是一致的。如果处理需要诸如在光束和光线之间共享堆栈之类的东西, 则可以使用一致的 $x/y/z$ 排序来获得接近 t 排序的性能。

[0233] 用于指示是否在返回时弹出遍历堆栈的光线标志 (例如, “noPopOnReturn”) 可指定是否要弹出堆栈, 和/或是否返回遍历的结果而不弹出堆栈。在不弹出遍历堆栈的情况下返回遍历的结果可以使 SM 能够在开始新的遍历之前重新运行相同的遍历或修改堆栈。

[0234] 控制碰撞报告的光线标志 (例如, “reportOnEnter” 标志) 可以指定 TTU 仅在 AABB 交点 t 大于或等于光线的 t_{min} 时报告子碰撞, 并且剔除 (和/或不报告至 SM) 其他。如果该交点 (在光线进入包围盒时) 发生在光线指定的感兴趣区域之前, 则该标志使得包围盒不会被报告给 SM, 即使它是相交的。该标志的一个示例性使用是用于光线行进, 其中在找到交点之后, 将 t_{min} 提前为该交点的起点。在重新启动时, 可能希望找到下一个交点, 但通常不希望再次报告刚刚返回的交点。通过设置 reportOnEnter 标志, 可以避免将交点再次返回到 SM, 因为重新启动的光线不会进入包围盒, 而是从其内部开始。

[0235] 控制是否在第一次碰撞时终止的光线标志 (例如, “terminateOnHit”) 指定 TTU 是否要在遍历期间发现的光线的第一次碰撞时返回, 或者继续遍历直到它可以返回发现的参数上最近的碰撞。

[0236] 指示哪些三角形被认为是面向前方的光线标志 (例如, “facingFrontFaceCW”) 可用于指定相交叶的某些处理。例如, 这些标志可以指定逆时针排列的三角形作为前向处理, 或者假设为右手坐标系则将顺时针排列的三角形作为前向处理。

[0237] 可以指定控制相交图元的剔除的光线标志 (例如, “cullMode”) 以指示没有剔除, 剔除后向图元, 剔除前向图元, 或者禁用剔除和图元测试。

[0238] 遍历逻辑 (例如, 遍历逻辑块 712) 基于 RayOp 测试的结果 (或者如果设置子项反转标识, 则为 RayOp 测试的结果的反转) 来执行由适当的一个或更多个模式标志枚举的动作。在上面的示例中, 由于 ch_f 模式标志指示在 RayOp 测试失败时要剔除子包围盒, 因此遍历逻辑不会将堆栈条目推送到该子包围盒的光线遍历堆栈上, 即使该光线也是可以与子包围盒相交, 并且相交的子包围盒的默认行为是将子包围盒推送到遍历堆栈中。请注意, 光线可以具有而不是指定 ch_f 模式标志的值, 该 ch_f 模式标志指示如果 RayOp 测试失败则要剔除子包围盒, 或者指示子包围盒将在 TTU 中遍历, 或者作为节点参考返回。

[0239] 可以以与光线的光线标志一致的方式执行遍历逻辑的动作。例如, 在光线标志指示特定遍历顺序的情况下, 根据 RayOp 测试选择的用于遍历的子包围盒可以以与由相应光线标志指定的遍历顺序一致的方式被推送到遍历堆栈。

[0240] 可以针对相交的包围盒的每个子包围盒重复步骤 1404-1414。当每个子节点或至少每个发现与光线相交的子节点都执行了 RayOp 时, 父包围盒已完成其遍历步骤。也就是说, 在 complet 仅包括根包围盒及其子包围盒的情况下, 该 complet 的遍历已经完成。更一般地说, 当 complet 包含根节点和一个以上级别的节点时, 当 complet 的所有叶节点或至少尚未被剔除的所有叶节点已经承受光线-包围盒相交测试和/或 RayOp 测试时, complet 的遍历就完成了。

[0241] 图 15 示出了当相交的包围盒是叶包围盒时可以在例如过程 1400 中的步骤 1414 执行的过程 1500 的流程图。

[0242] 在步骤 1502, 接收关于光线-包围盒相交检测的信息。可以基于相交的叶包围盒中

包含的场景几何体的类型来确定下一步骤。在步骤1504、1508和1512中分别处理实例节点、项目范围和图元范围。

[0243] 如果相交的包围盒中的场景几何体包括实例节点,则在步骤1504,执行光线-实例节点相交检测。如果确定光线与实例节点相交,则使用以下信息将条目推送到遍历堆栈:实例节点的地址,其指定从当前(例如,世界)坐标空间到不同坐标空间的变换,以及指向另一个(例如,对象)坐标空间中另一个BVH的根的指针,指示是否将查询返回到SM的“returnToSM”标志,以便SM可以在当前BVH中保存查询的遍历状态并应用变换,以及“初始化”标志,其指示是否已经初始化了光线(或从世界空间光线克隆的光线的另一个实例)。

[0244] 如果相交的包围盒中的场景几何体包括项目范围,则在步骤1506,执行光线-项目范围相交检测。基于相交检测,通过以下信息将条目推送到遍历堆栈:指定项目范围数据的起始地址的起始地址,以及指示该范围中的项目数的计数。

[0245] 如果相交的包围盒中的场景几何包括图元范围,则在步骤1512执行光线-图元范围相交检测。在步骤1514,确定是否处理TTU中的图元/图元范围,或者利用以下信息将条目推入遍历堆栈:图元范围中的最后一个块的地址(“addrLast”),第一个高速缓存行中图元范围的第一个图元的索引(“triIdx”),该范围中最后一个高速缓存行中最后一个图元的索引,或者表示该范围在最后一个高速缓存行(“triEnd”)中结束的零,包含图元范围的块数(“行”),指示是否要剔除不透明图元的“coflag”,指示是否要剔除 α 图元的“caflag”,以及“rp flag”,其指示是否通过了图元范围RayOp测试。

[0246] 在RayOp和模式标志指示将结果返回到流式处理器以进行特殊处理的情况下,光线、其遍历状态及其结果队列内容全部返回到流式处理器。流式处理器可以执行可能需要的任何附加处理(例如,内插纹理坐标并评估 α 纹理以确定是否要忽略 α 交点)并且如果需要,流式处理器可以采用相同或更改的查询参数重新提交查询(例如,通过拉近tmax重新提交时可缩短光线)。

[0247] 图16示出了当例如在步骤1502的操作中,相对于包括可以在TTU中解决的图元范围(例如,三角形范围)叶包围盒检测到光线-包围盒交点1602时可以执行的过程1600的流程图。

[0248] 在步骤1604,相对于包围盒执行光线的RayOp测试。当RCT单元710与参考图元范围的子包围盒相交时,TL子单元712执行由使用ray的A,B参数和complet子项的rval参数的光线的RayOp查询参数指定的RayOp测试,并将RayOp测试的结果存储在存储在堆栈条目中对应于被推送到遍历堆栈顶部的图元范围的rp标志中,如上面的操作1518所述。

[0249] 当堆栈管理单元740从堆栈顶部弹出该条目并向图元调度器发送激活时,rp标志以及该激活由图元调度器转发到RPT单元720。当RPT 720检测到不透明交点或 α 交点时,RPT 720检索对应于图元碰撞类型和rp标志的一个或更多个模式标志,并应用适当的动作来处理交点。例如,如果具有rpflag=TRUE的光线与不透明图元相交,并且光线的“op_p”标志指示来自通过RayOp测试的不透明图元碰撞将被剔除,则RPT 720将不会将不透明交点推送到结果队列。

[0250] 在步骤1606,将RayOp测试的结果存储在rp标志中,并且将包括rp标志的条目推送到遍历堆栈。

[0251] 在步骤1608,弹出堆栈并且将rp标志提供给光线-图元测试块(例如,图10B中的光

线/三角形测试1026)。

[0252] 当堆栈管理单元从堆栈顶部弹出该条目并向图元调度器发送激活时,图元调度器将rp标志与该激活一起转发到RPT单元。

[0253] 在步骤1610,在光线-图元测试中,检测光线和图元之间的交点。当RPT检测到不透明的交点或 α 交点时,RPT检索对应于图元碰撞类型和rp标志的一个或多个模式标志,并应用适当的动作来处理交点。

[0254] 作为完成的光线-图元测试的结果,RPT模式标志指定TTU行为。示例RPT模式标志“at_f”(“modeAlphaFail”)指定在相交的 α 图元的RayOp测试失败时要执行的操作。支持的动作可以包括将 α 碰撞推送到结果队列(例如,结果1310)以传送到SM,剔除 α 碰撞(抑制 α 图元测试),或者将 α 碰撞推送到结果队列作为不透明碰撞。另一个示例RPT模式标志包括模式标志“at_p”(“modeAlphaPass”),其指定当针对相交的 α 图元的RayOp测试通过时要执行的动作。支持的动作可能包括将 α 图元数据推送到结果队列,剔除 α 碰撞,以及将 α 碰撞推送到结果队列作为不透明碰撞。

[0255] 模式标志“ot_f”(“modeOpaqueFail”)指定当针对相交的不透明图元的RayOp测试失败时要执行的动作。支持的操作可能包括将不透明碰撞推送到结果队列,剔除不透明碰撞,以及将不透明碰撞推送到结果队列作为 α 碰撞。模式标志“op_p”(“modeAlphaPass”)指定当针对相交的不透明图元的RayOp测试通过时要执行的动作。支持的动作可包括将RayOp测试推送到结果队列,剔除不透明碰撞,以及将不透明碰撞推送到结果队列作为 α 碰撞。

[0256] 一些节点或叶(例如,实例节点、图元范围)还可以包括与RayOp测试和模式标志交互和/或覆写RayOp测试和模式标志的标志。例如,实例节点报头具有标志“foa”(强制不透明为 α),“fao”(强制 α 为不透明),“fnc”(强制无剔除)可以覆写对象空间光线的模式标志“op_p”,“op_f”,“ap_f”,“ap_p”的值,用于覆写从相应的世界空间光线复制的值。在示例性实施例中,用于较低bvh遍历的光线模式位从较高bvh光线模式位复制。

[0257] 在步骤1612,基于rp标志和光线图元模式标志选择动作。例如,如果具有rp flag = TRUE的光线与不透明图元相交,并且光线的“op_p”标志指示来自通过RayOp测试的不透明图元碰撞将被剔除,然后RPT不会将不透明交点推送到结果队列。动作的选择还可以考虑在测试的图元和/或其他光线标志中指定的任何覆写参数。

[0258] 如上所述,示例性实施例提供了通过查询以编程方式改变加速结构的遍历的默认行为的能力。在一些示例性实施例中,在芯片区域约束和在遍历的硬件加速上施加的其他硬件约束内启用这些能力。除了其他方面之外,实施例实现的能力提供了应用程序以选择性地处理在某些深度级别的对象的几何图元以用于光线追踪,以在光线追踪期间在BVH中的对象的多个版本中进行选择,根据几何细节水平选择性地处理对象的几何图元,根据分辨率水平选择性地处理对象的几何图元,根据材料类型选择性地处理对象的几何图元,根据它们相交的光线距离选择性地处理对象的几何图元等。示例性实施例还可以在每光线的基础上执行诸如背面剔除之类的操作,其提供忽略指向远离特定光线的原点的所有图元,以及正面剔除,其提供忽略指向特定光线原点的所有图元。此外,一些示例性实施例还提供用于特定交点的覆写每光线行为:例如,BVH的一些节点(诸如实例节点)可以具有覆写每个光线行为的覆写标记,例如在实例基础上背面剔除,以便对于对象(诸如叶子),其中单个三角形图元表示叶子,无论三角形是否向后,光线都与它相交。

[0259] 尽管关于作为加速结构的BVH和作为查询的光线描述了示例性实施例,但是实施例不限于此。例如,在一些其他实施例中,查询可以包括波束或其他几何形状而不是光线,和/或与上述BVH不同的层次结构可以用作加速结构。本领域技术人员还将理解特定模式标志、RayOp操作码、光线参数值、rval参数是示例,并且实施例可以包括那些和/或其他参数的任何组合以提供加速结构的遍历的查询特定行为修改。

[0260] 上述能力可以在并行处理单元中实现,或者与并行处理单元相关联,例如下面结合图17-图23描述的并行处理单元。

[0261] 更详细的TTU和GPU架构

[0262] 现在将根据用户的期望,阐述关于可以或不实现前述框架的各种可选架构和特征的更多说明性信息。应该特别注意的是,出于说明性目的阐述了以下信息,并且不应该被解释为以任何方式进行限制。任选的以下特征可以任选地并入或不排除所描述的其他特征。

[0263] 示例性并行处理架构

[0264] 图17示出了根据一个实施例的并行处理单元(PPU) 1700。在一个实施例中,PPU 1700是在一个或多个集成电路器件上实现的多线程处理器。PPU 1700是设计用于并行处理许多线程的延迟隐藏架构。线程(即,执行线程)是,被配置为由PPU 1700执行的指令集的实例。在一个实施例中,PPU 1700是图形处理单元(例如,诸如关于图1描述的GPU),其被配置为实现用于处理三维(3D)图形数据的图形渲染管线,以便生成用于在显示设备(诸如液晶显示(LCD)设备、有机发光二极管(OLED)设备、透明发光二极管(TOLED)设备、场发射显示器(FED)、场顺序显示器、投影显示器、头戴式显示器或任何其他所需显示器)上显示的二维(2D)图像数据。在其他实施例中,PPU 1700可以用于执行通用计算。尽管为了说明的目的本文提供了一个示例性并行处理器,但应特别指出的是,该处理器仅出于说明目的进行阐述,并且可使用任何处理器来补充和/或替代该处理器。

[0265] 例如,一个或多个PPU 1700可以被配置为加速数千个高性能计算(HPC)、数据中心和机器学习应用。PPU 1700可被配置为加速众多深度学习系统和应用,包括自动驾驶汽车平台、深度学习、高精度语音、图像和文本识别系统、智能视频分析、分子模拟、药物发现、疾病诊断、天气预报、大数据分析、天文学、分子动力学模拟、金融建模、机器人技术、工厂自动化、实时语言翻译、在线搜索优化和个性化用户推荐等。

[0266] 如图17所示,PPU 1700包括输入/输出(I/O)单元1705、前端单元1715、调度器单元1720、工作分配单元1725、集线器1730、交叉开关(Xbar) 1770、一个或多个通用处理集群(GPC) 1750以及一个或多个分区单元1780。PPU 1700可以经由一个或多个高速NVLink 1710互连连接到主机处理器或其他PPU 1700。PPU 1700可以经由互连1702连接到主机处理器或其他外围设备。PPU 1700还可以连接到包括多个存储器设备1704的本地存储器。在一个实施例中,本地存储器可以包括多个动态随机存取存储器(DRAM)器件。DRAM器件可以被配置为高带宽存储器(HBM)子系统,其中多个DRAM裸晶(die)堆叠在每个器件内。

[0267] NVLink 1710互连使得系统能够扩展并且包括与一个或多个CPU结合的一个或多个PPU 1700,支持PPU 1700和CPU之间的高速缓存一致性以及CPU主控。数据和/或命令可以由NVLink 1710通过集线器1730发送到PPU 1700的其他单元或从其发送,例如一个或多个复制引擎、视频编码器、视频解码器、电源管理单元等(未明确示出)。结合图22更详

细地描述NVLink 1710。

[0268] I/O单元1705被配置为通过互连1702从主机处理器(未示出)发送和接收通信(即,命令、数据等)。I/O单元1705可以经由互连1702直接与主机处理器通信,或通过一个或更多个中间设备(诸如存储器桥)与主机处理器通信。在一个实施例中,I/O单元1705可以经由互连1702与一个或更多个其他处理器(例如,一个或更多个PPU 1700)通信。在一个实施例中,I/O单元1705实现外围组件互连高速(PCIe)接口,用于通过PCIe总线进行通信,并且互连1702是PCIe总线。在替代实施例中,I/O单元1705可以实现其他类型的已知接口,用于与外部设备进行通信。

[0269] I/O单元1705对经由互连1702接收的分组进行解码。在一个实施例中,分组表示被配置为使PPU 1700执行各种操作的命令。I/O单元1705按照命令指定将解码的命令发送到PPU 1700的各种其他单元。例如,一些命令可以被发送到前端单元1715。其他命令可以被发送到集线器1730或PPU 1700的其他单元,诸如一个或更多个复制引擎、视频编码器、视频解码器、电源管理单元等(未明确示出)。换句话说,I/O单元1705被配置为在PPU 1700的各种逻辑单元之间和之中路由通信。

[0270] 在一个实施例中,由主机处理器执行的程序在缓冲区中对命令流进行编码,该缓冲区向PPU 1700提供工作量用于处理。工作量可以包括要由那些指令处理的若干指令和数据。缓冲区是存储器中可由主机处理器和PPU 1700两者访问(即,读/写)的区域。例如,I/O单元1705可以被配置为经由通过互连1702发送的存储器请求访问连接到互连1702的系统存储器中的缓冲区。在一个实施例中,主机处理器将命令流写入缓冲区,然后向PPU 1700发送指向命令流开始的指针。前端单元1715接收指向一个或更多个命令流的指针。前端单元1715管理一个或更多个流,从流读取命令并将命令转发到PPU 1700的各个单元。

[0271] 前端单元1715耦合到调度器单元1720,调度器单元1720配置各种GPC 1750以处理由一个或更多个流定义的任务。调度器单元1720被配置为追踪与由调度器单元1720管理的各种任务相关的状态信息。状态可以指示任务被指派给哪个GPC 1750,该任务是活动的还是不活动的,与该任务相关联的优先级等等。调度器单元1720管理一个或更多个GPC 1750上的多个任务的执行。

[0272] 调度器单元1720耦合到工作分配单元1725,工作分配单元1725被配置为分派任务以在GPC 1750上执行。工作分配单元1725可以追踪从调度器单元1720接收到的多个调度任务。在一个实施例中,工作分配单元1725为每个GPC 1750管理待处理(pending)任务池和活动任务池。待处理任务池可以包括多个槽(例如,32个槽),其包含被指派为由特定GPC 1750处理的任務。活动任务池可以包括多个槽(例如,4个槽),用于正在由GPC 1750主动处理的任務。当GPC 1750完成任务的执行时,该任务从GPC 1750的活动任务池中逐出,并且来自待处理任务池的其他任务之一被选择和调度以在GPC 1750上执行。如果GPC 1750上的活动任务已经空闲,例如在等待数据依赖性被解决时,那么活动任务可以从GPC 1750中逐出并返回到待处理任务池,而待处理任务池中的另一个任务被选择并调度以在GPC 1750上执行。

[0273] 工作分配单元1725经由XBar(交叉开关)1770与一个或更多个GPC 1750通信。XBar 1770是将PPU 1700的许多单元耦合到PPU 1700的其他单元的互连网络。例如,XBar 1770可以被配置为将工作分配单元1725耦合到特定的GPC 1750。虽然没有明确示出,但PPU 1700的一个或更多个其他单元也可以经由集线器1730连接到XBar 1770。

[0274] 任务由调度器单元1720管理并由工作分配单元1725分派给GPC 1750。GPC 1750被配置为处理任务并生成结果。结果可以由GPC 1750内的其他任务消耗,经由XBar 1770路由到不同的GPC 1750,或者存储在存储器1704中。结果可以经由分区单元1780写入存储器1704,分区单元1780实现用于从存储器1704读取数据和/或向存储器1704写入数据的存储器接口。结果可以经由NVLink1710发送到另一个PPU 1704或CPU。在一个实施例中,PPU 1700包括数目为U的分区单元1780,其等于耦合到PPU 1700的独立且不同的存储器设备1704的数目。下面将结合图18更详细地描述分区单元1780。

[0275] 在一个实施例中,主机处理器执行实现应用程序编程接口(API)的驱动程序内核,其使得能够在主机处理器上执行一个或更多个应用程序,以调度用于在PPU 1700上执行的操作。在一个实施例中,多个计算机应用程序由PPU 1700同时执行,并且PPU 1700为多个计算机应用程序提供隔离、服务质量(QoS)和独立地址空间。应用程序可以生成指令(即API调用),其使得驱动程序内核生成一个或更多个任务以由PPU 1700执行。驱动程序内核将任务输出到正在由PPU 1700处理的一个或更多个流。每个任务可以包括一个或更多个相关线程组,本文称为线程束(warp)。在一个实施例中,线程束包括可以并行执行的32个相关线程。协作线程可以指代包括用于执行任务的指令并且可以通过共享存储器交换数据的多个线程。结合图20更详细地描述线程和协作线程。

[0276] 图19示出了根据一个实施例的图17的PPU 1700的GPC 1750。如图19所示,每个GPC 1750包括用于处理任务的多个硬件单元。在一个实施例中,每个GPC 1750包括管线管理器1810、预光栅操作单元(PROP)1815、光栅引擎1825、工作分配交叉开关(WDX)1880、存储器管理单元(MMU)1890以及一个或更多个数据处理集群(DPC)1820。应当理解,图19的GPC 1750可以包括代替图19中所示的单元或者除了图19中所示的单元之外的其他硬件单元。

[0277] 在一个实施例中,GPC 1750的操作由管线管理器1810控制。管线管理器1810管理用于处理分配给GPC 1750的任务的一个或更多个DPC 1820的配置。在一个实施例中,管线管理器1810可以配置一个或更多个DPC 1820中的至少一个来实现图形渲染管线的至少一部分。例如,DPC 1820可以被配置为在可编程流式多处理器(SM)1840上执行顶点着色器程序。管线管理器1810还可以被配置为将从工作分配单元1725接收的分组路由到GPC 1750中的适当逻辑单元。例如,一些分组可以被路由到PROP 1815和/或光栅引擎1825中的固定功能硬件单元,而其他分组可以被路由到DPC 1820以供图元引擎1835或SM 1840处理。在一个实施例中,管线管理器1810可以配置一个或更多个DPC 1820中的至少一个以实现神经网络模型和/或计算管线。

[0278] PROP单元1815被配置为将由光栅引擎1825和DPC 1820生成的数据路由到光栅操作(ROP)单元,其结合图18更详细地描述。PROP单元1815还可以被配置为执行颜色混合的优化,组织像素数据,执行地址转换等。

[0279] 光栅引擎1825包括被配置为执行各种光栅操作的多个固定功能硬件单元。在一个实施例中,光栅引擎1825包括设置引擎、粗光栅引擎、剔除引擎、裁剪引擎、精细光栅引擎和图块聚合引擎。设置引擎接收变换后的顶点并生成与由顶点定义的几何图元相关联的平面方程。平面方程被发送到粗光栅引擎以生成图元的覆盖信息(例如,图块的x、y覆盖掩码)。粗光栅引擎的输出被发送到剔除引擎,其中与未通过z-测试的图元相关联的片段被剔除,并且未被剔除的片段被发送到裁剪引擎,其中位于视锥体之外的片段被裁剪掉。那些经过

裁剪和剔除后留下来的片段可以被传递到精细光栅引擎,以基于由设置引擎生成的平面方程生成像素片段的属性。光栅引擎1825的输出包括例如要由在DPC 1820内实现的片段着色器处理的片段。

[0280] 包括在GPC 1750中的每个DPC 1820包括M管道控制器(MPC)1830、图元引擎1835、一个或更多个SM 1840、一个或更多个纹理单元1842以及一个或更多个TTU 1844。TTU 1844可以与上述TTU 700类似地操作。MPC 1830控制DPC 1820的操作,将从管线管理器1810接收的分组路由到DPC 1820中的适当单元。例如,与顶点相关联的分组可以被路由到图元引擎1835,其被配置为获取与来自存储器1704的顶点相关联的顶点属性。相反,与着色器程序相关联的分组可以被发送到SM 1840。

[0281] SM 1840包括被配置为处理由多个线程表示的任务的可编程流式处理器。每个SM 1840是多线程的并且被配置为同时执行来自特定线程组的多个线程(例如,32个线程)。在一个实施例中,SM 1840实现SIMD(单指令、多数据)体系架构,其中线程组(即,warp)中的每个线程被配置为基于相同的指令集来处理不同的数据集。线程组中的所有线程都执行相同的指令。在另一个实施例中,SM 1840实现SMT(单指令、多线程)体系架构,其中线程组中的每个线程被配置为基于相同的指令集处理不同的数据集,但是其中线程组中的各个线程在执行期间被允许发散。在一个实施例中,为每个线程束维护程序计数器、调用栈和执行状态,当线程束内的线程发散时,使能线程束和线程束中的串行执行之间的并发。在另一个实施例中,为每个个体线程维护程序计数器、调用栈和执行状态,使能在线程束内和线程束之间的所有线程之间的相等并发。当为每个个体线程维护执行状态时,执行相同指令的线程可以收敛并且并行执行以获得最大效率。下面将结合图20和图21更详细地描述SM 1840。

[0282] MMU 1890提供GPC 1750和分区单元1780之间的接口。MMU 1890可以提供虚拟地址到物理地址的变换,存储器保护和存储器请求的仲裁。在一个实施例中,MMU 1890提供一个或更多个转换后备缓冲器(TLB),用于执行虚拟地址到存储器1704中的物理地址的变换。

[0283] 图18示出了根据实施例的图17的PPU 1700的存储器分区单元1780。如图18所示,存储器分区单元1780包括光栅操作(ROP)单元1850、二级(L2)高速缓存1860和存储器接口1870。存储器接口1870耦合到存储器1704。存储器接口1870可以实现用于高速数据传输的32、64、128、1024位数据总线等。在一个实施例中,PPU 1700包含U个存储器接口1870,每对分区单元1780一个存储器接口1870,其中每对分区单元1780连接到相应的存储器设备1704。例如,PPU 1700可以连接到多达Y个存储器设备1704,例如高带宽存储器堆栈或图形双数据速率,版本5,同步动态随机存取存储器或其他类型的持久存储器。

[0284] 在一个实施例中,存储器接口1870实现HBM2存储器接口并且Y等于U的一半。在一个实施例中,HBM2存储器堆栈与PPU 1700位于相同的物理封装上,与传统的GDDR5SDRAM系统相比,其提供了相当大的功率和面积节省。在一个实施例中,每个HBM2堆栈包括四个存储器裸晶并且Y等于4,HBM2堆栈包括每个裸晶两个128位通道,总共8个通道和1024位的数据总线宽度。

[0285] 在一个实施例中,存储器1704支持单纠错双错误检测(SECDED)纠错码(ECC)以保护数据。ECC为对数据损坏敏感的计算应用程序提供更高的可靠性。可靠性在大规模集群计算环境中尤其重要,其中PPU 1700处理非常大的数据集和/或长时间运行应用程序。

[0286] 在一个实施例中,PPU 1700实现多级存储器层次。在一个实施例中,存储器分区单

元1780支持统一存储器,以为CPU和PPU 1700存储器提供单个统一虚拟地址空间,从而实现虚拟存储器系统之间的数据共享。在一个实施例中,追踪PPU 1700对位于其他处理器上的存储器的访问频率,以确保将存储器页面移动到更频繁地访问页面的PPU 1700的物理存储器。在一个实施例中,NVLink 1710支持地址转换服务,允许PPU 1700直接访问CPU的页表并由PPU 1700提供对CPU存储器的完全访问。

[0287] 在一个实施例中,复制引擎在多个PPU 1700之间或在PPU 1700和CPU之间传输数据。复制引擎可以为未映射到页表的地址生成页面错误。然后,存储器分区单元1780可以服务页面错误,将地址映射到页表中,之后复制引擎可以执行该传输。在传统系统中,存储器被固定(即,不可分页)以用于多个处理器之间的多个复制引擎操作,从而大大减少了可用存储器。使用硬件页面错误,可以将地址传递给复制引擎,而不必担心存储器页面是否驻留,并且复制过程是透明的。

[0288] 来自存储器1704或其他系统存储器的数据可以由存储器分区单元1780获取并存储在L2高速缓存1860中,L2高速缓存1860位于芯片上并且在各种GPC 1750之间共享。如图所示,每个存储器分区单元1780包括与相应的存储器设备1704相关联的L2高速缓存1860的一部分。然后,可以在GPC 1750内的各种单元中实现较低级的高速缓存。例如,SM 1840中的每一个可以实现一级(L1)高速缓存。L1高速缓存是专用于特定SM 1840的专用存储器。可以获得来自L2高速缓存1860的数据并将其存储在每个L1高速缓存中,以便在SM 1840的功能单元中进行处理。L2高速缓存1860被耦合到存储器接口1870和XBar 1770。

[0289] ROP单元1850执行与像素颜色有关的图形光栅操作,例如颜色压缩、像素混合等。ROP单元1850还结合光栅引擎1825实现深度测试,接收与来自光栅引擎1825的剔除引擎的像素片段相关联的样本位置的深度。针对深度缓冲器中的对应深度测试与片段关联的样本位置的深度。如果片段通过样本位置的深度测试,则ROP单元1850更新深度缓冲器并将深度测试的结果发送到光栅引擎1825。应当理解,分区单元1780的数量可以不同于GPC 1750的数量,并且因此每个ROP单元1850可以耦合到每个GPC 1750。ROP单元1850追踪从不同GPC 1750接收的分组,并且确定由ROP单元1850生成的结果通过Xbar 1770被路由到哪个GPC 1750。虽然ROP单元1850被包括在图18中的存储器分区单元1780内,但在其他实施例中,ROP单元1850可以在存储器分区单元1780之外。例如,ROP单元1850可以驻留在GPC 1750或另一单元中。

[0290] 示例性流式多处理器

[0291] SM 1840包括被配置为处理由多个线程表示的任务的可编程流式处理器。每个SM 1840是多线程的并且被配置为同时执行来自特定线程组的多个线程(例如,32个线程,其包括线程束)。在一个实施例中,SM 1840实现SIMD(单指令、多数据)体系架构,其中线程组(即,warp)中的每个线程被配置为基于相同的指令集来处理不同的数据集。线程组中的所有线程都执行相同的指令。在另一个实施例中,SM 1840实现SMT(单指令、多线程)体系架构,其中线程组中的每个线程被配置为基于相同的指令集处理不同的数据集,但是其中线程组中的各个线程在执行期间被允许发散。在一个实施例中,为每个线程束维护程序计数器、调用栈和执行状态,当线程束内的线程发散时,使能线程束和线程束中的串行执行之间的并发。在另一个实施例中,为每个个体线程维护程序计数器、调用栈和执行状态,使能在线程束内和线程束之间的所有线程之间的相等并发。当为每个个体线程维护执行状态时,

执行相同指令的线程可以收敛并且并行执行以获得最大效率。

[0292] 图20示出了根据一个实施例的图19的流式多处理器1840。如图20所示,SM 1840包括指令高速缓存1905、一个或更多个调度器单元1910、寄存器文件1920、一个或更多个处理核心1950、一个或更多个特殊功能单元(SFU) 1952、一个或更多个加载/存储单元(LSU) 1954、互连网络1980、共享存储器/L1高速缓存1970。

[0293] 如上所述,工作分配单元1725分派任务以在PPU 1700的GPC 1750上执行。任务被分配给GPC 1750内的特定DPC 1820,并且如果任务与着色器程序相关联,则该任务可以被分配给SM 1840。调度器单元1910从工作分配单元1725接收任务并且管理指派给SM 1840的一个或更多个线程块的指令调度。调度器单元1910调度线程块以作为并行线程的线程束执行,其中每个线程块被分配至少一个线程束。在一个实施例中,每个线程束执行32个线程。调度器单元1910可以管理多个不同的线程块,将线程束分配给不同的线程块,然后在每个时钟周期期间将来自多个不同的协作组的指令分派到各个功能单元(即,核心1950、SFU 1952和LSU 1954)。

[0294] 协作组是用于组织通信线程组的编程模型,其允许开发者表达线程正在通信所采用的粒度,使得能够表达更丰富、更高效的并行分解。协作启动API支持线程块之间的同步性,以执行并行算法。常规的编程模型为同步协作线程提供了单一的简单结构:跨线程块的所有线程的屏障(barrier)(即,syncthreads()函数)。然而,程序员通常希望以小于线程块粒度的粒度定义线程组,并在所定义的组内同步,以集体的全组功能接口(collective group-wide function interface)的形式使能更高的性能、设计灵活性和软件重用。

[0295] 协作组使得程序员能够在子块(即,像单个线程一样小)和多块粒度处明确定义线程组并且执行集体操作,诸如协作组中的线程上的同步性。编程模型支持跨软件边界的干净组合,以便库和效用函数可以在本地环境中安全地同步,而无需对收敛进行假设。协作组图元启用合作伙伴并行的新模式,包括生产者-消费者并行、机会主义并行以及跨整个线程块网格的全局同步。

[0296] 分派单元1915被配置为向一个或更多个功能单元发送指令。在该实施例中,调度器单元1910包括两个分派单元1915,其使得能够在每个时钟周期期间调度来自相同线程束的两个不同指令。在替代实施例中,每个调度器单元1910可以包括单个分派单元1915或附加分派单元1915。

[0297] 每个SM 1840包括寄存器文件1920,其提供用于SM 1840的功能单元的一组寄存器。在一个实施例中,寄存器文件1920在每个功能单元之间被划分,使得每个功能单元被分配寄存器文件1920的专用部分。在另一个实施例中,寄存器文件1920在由SM 1840执行的不同线程束之间被划分。寄存器文件1920为连接到功能单元的数据路径的操作数提供临时存储。图21示出了SM 1840中的寄存器文件的示例性配置。

[0298] 每个SM 1840包括L个处理核心1950。在一个实施例中,SM 1840包括大量(例如128个等)不同的处理核心1950。每个核心1950可以包括完全管线化的、单精度、双精度和/或混合精度处理单元,其包括浮点运算逻辑单元和整数运算逻辑单元。在一个实施例中,浮点运算逻辑单元实现用于浮点运算的IEEE 754-2008标准。在一个实施例中,核心1950包括64个单精度(32位)浮点核心、64个整数核心、32个双精度(64位)浮点核心和8个张量核心(tensor core)。

[0299] 张量核心被配置为执行矩阵运算,并且在一个实施例中,一个或更多个张量核心被包括在核心1950中。具体地,张量核心被配置为执行深度学习矩阵运算,诸如用于神经网络训练和推理的卷积运算。在一个实施例中,每个张量核心在 4×4 矩阵上运算并且执行矩阵乘法和累加运算 $D=A \times B+C$,其中A、B、C和D是 4×4 矩阵。

[0300] 在一个实施例中,矩阵乘法输入A和B是16位浮点矩阵,而累加矩阵C和D可以是16位浮点或32位浮点矩阵。张量核心对16位浮点输入数据进行运算并与32位浮点进行累加。16位浮点乘法需要64次运算,产生全精度的积,然后使用32位浮点与 $4 \times 4 \times 4$ 矩阵乘法的其他中间积相加来累加。在实践中,张量核心用于执行由这些较小的元素建立的更大的二维或更高维的矩阵运算。API (诸如CUDA 9C++API) 公开了专门的矩阵加载、矩阵乘法和累加以及矩阵存储运算,以便有效地使用来自CUDA-C++程序的张量核心。在CUDA层面,线程束级接口假定 16×16 尺寸矩阵跨越线程束的全部32个线程。

[0301] 每个SM 1840还包括执行特殊函数(例如,属性评估、倒数平方根等)的M个SFU 1952。在一个实施例中,SFU 1952可以包括树遍历单元,其被配置为遍历层次树形数据结构。在一个实施例中,SFU 1952可以包括被配置为执行纹理映射过滤操作的纹理单元。在一个实施例中,纹理单元被配置为从存储器1704加载纹理映射(例如,纹理像素的2D阵列)并且对纹理映射进行采样以产生经采样的纹理值,用于在由SM 1840执行的着色器程序中使用。在一个实施例中,纹理映射被存储在共享存储器/L1高速缓存1970中。纹理单元实现纹理操作,诸如使用mip映射(即,不同细节层次的纹理映射)的过滤操作。在一个实施例中,每个SM 1740包括两个纹理单元。

[0302] 每个SM 1840还包括N个LSU 1954,其实现共享存储器/L1高速缓存1970和寄存器文件1920之间的加载和存储操作。每个SM 1840包括互连网络1980,其将每个功能单元连接到寄存器文件1920以及将LSU 1954连接到寄存器文件1920、共享存储器/L1高速缓存1970。在一个实施例中,互连网络1980是交叉开关,其可以被配置为将任何功能单元连接到寄存器文件1920中的任何寄存器,以及将LSU 1954连接到寄存器文件和共享存储器/L1高速缓存1970中的存储器位置。

[0303] 共享存储器/L1高速缓存1970是片上存储器阵列,其允许数据存储和SM 1840与图元引擎1835之间以及SM 1840中的线程之间的通信。在一个实施例中,共享存储器/L1高速缓存1970包括128KB的存储容量并且在从SM 1840到分区单元380的路径中。共享存储器/L1高速缓存1970可以用于高速缓存读取和写入。共享存储器/L1高速缓存1970、L2高速缓存1860和存储器1704中的一个或更多个是后备存储。

[0304] 将数据高速缓存和共享存储器功能组合成单个存储器块为两种类型的存储器访问提供了最佳的总体性能。该容量可由不使用共享存储器的程序用作高速缓存。例如,如果将共享存储器配置为使用一半容量,则纹理和加载/存储操作可以使用剩余容量。在共享存储器/L1高速缓存1970内的集成使共享存储器/L1高速缓存1970起到用于流式传输数据的高吞吐量管道的作用,并且同时提供高带宽和对频繁重用数据的低延迟的访问。

[0305] 图21示出了SM 1840的一个示例性架构。如图19所示,SM 1840可以耦合到一个或更多个纹理单元1842和/或一个或更多个TTU 700。作为性能和区域之间的折衷,一个示例性非限制性实施例可以包括单个纹理单元1842和/或每组SM 1840单个TTU 700(例如,参见图19)。TTU 700可以经由存储器输入-输出中的TTU输入/输出块与SM 1840通信,并且经由

专用读取接口与L1高速缓存通信。在一个示例性实施例中,TTU 700仅从主存储器读取并且不写入主存储器。

[0306] 示例性更详细的TTU架构

[0307] 如上所述,TTU 700可以是SM 1840的协处理器。与纹理处理器类似,它通过一组SM指令暴露,访问存储器作为L1高速缓存的只读客户端,并返回结果进入SM寄存器文件。与一些纹理处理器不同,对于典型查询,可能需要传入和传出TTU 700的数据量使得在一些实施例中难以在单个指令中指定所有源寄存器和目标寄存器(并且因为大多数这样的数据是每个线程唯一的,没有纹理头和采样器的TTU模拟)。结果,在一些实施例中,TTU 700通过多指令序列编程。在一些实现中,该序列可以概念化为单个“宏指令”。

[0308] 同样像纹理单元1842,在一些实现中,TTU 700可以依赖于由软件预先填充的存储器中的某些只读数据结构。这些包括:

[0309] --一个或更多个BVH,其中每个BVH例如是轴对齐的包围盒的树,以压缩格式存储,与未压缩的表示相比,其大大减少了存储器通信量。BVH中的每个节点都存储为complet结构,其中一些实现中的大小和对齐与L1高速缓存行的大小和对齐相匹配。优选地,将给定父级的子级complet连续地存储在存储器中,并且以压缩形式存储子指针。

[0310] --零个或更多个实例节点,其提供将一个BVH的叶连接到另一个BVH的根的方式。实例节点可以是也对齐的数据结构。该结构可以包含指向子BVH的指针,影响子BVH中的背面剔除行为的标记,以及对应于从顶层BVH(通常为“世界空间”)的坐标系到子BVH(通常为“对象空间”)的坐标系的任意变换矩阵(在齐次坐标中)的前三行的矩阵。在一些实施例中,矩阵的最后一行在一些实现中隐含地为(0,0,0,1)。

[0311] --零个或更多个三角形或其他图元缓冲区,包含例如存储为每顶点坐标的三元组或以TTU 700理解的无损压缩格式存储的三角形。此外,可以为每个三角形或其他图元提供 α 位,指示需要由软件进行特殊处理的三角形,以确定三角形实际上是否与给定光线相交。三角形缓冲区可以组织成块。也可能存在每个三角形力-无-剔除功能位。设置时,该位表示三角形的两侧应被视为相对于剔除的前向或后向,即,不应该剔除三角形,因为光线与“后”而不是“前”相交。最简单的用例是用于表示叶的单个三角形,如果光线在背面上碰撞它,我们仍然可以看到叶。

[0312] 在一些实施例中,TTU 700是无状态的,意味着在查询之间不在TTU中维持架构状态。同时,在SM 1840上运行软件对要求继续先前的查询通常是有用的,这意味着相关状态应该由TTU 700写入寄存器,然后在寄存器中传递回TTU(通常原地)继续。该状态可以采用遍历堆栈的形式,其追踪BVH遍历的进展。

[0313] 还可以提供少量堆栈初始化器以开始给定类型的新查询,例如:

[0314] • 从complet开始遍历

[0315] • 光线与一系列三角形的交点

[0316] • 光线与一系列三角形的交点,然后从complet开始遍历

[0317] • 从三角形缓冲区中获取给定三角形的顶点

[0318] • 可选支持在“从complet开始的遍历”和“光线与一系列三角形的交点”前面的实例变换。

[0319] 顶点获取是一种简单查询,其可以由包括堆栈初始化器的请求数据指定而不是其

他任何东西。其他查询类型可能需要指定光线或光束,以及堆栈或堆栈初始化器以及描述查询详细信息的各种光线标记。光线由其三坐标原点、三坐标方向以及沿光线的 t 参数的最小值和最大值给定。另外通过第二原点和方向给定光束。

[0320] 各种光线标记可用于控制遍历行为、背面剔除和各种子节点类型的处理的各个方面,受制于可选的rayOp测试的通过/失败状态。RayOps为TTU的容量增加了相当大的灵活性。在一些示例性实施例中,RayOps部分引入两个光线标记版本,其可以基于对光线传送的数据的指定操作和存储在complet中的数据来动态选择。为了探索这样的标记,首先要了解BVH中允许的不同类型的子节点,以及TTU 700可以返回SM的各种碰撞类型。示例性节点类型是:

[0321] ■子complet(即,内部节点)

[0322] 默认情况下,TTU 700通过下降到子complet继续遍历。

[0323] ■三角形范围,对应于三角形缓冲区内的一组连续三角形

[0324] (1)默认情况下,通过三角形相交测试并相应地缩短光线,由TTU 700本地处理光线遇到三角形范围。如果遍历完成并且三角形被碰撞,则默认行为是将三角形ID以及交点的 t 值和重心坐标返回到SM 1840。这是“三角”碰撞类型。

[0325] (2)默认情况下,即使遍历尚未完成,设置了 α 位的相交三角形也会返回到SM 1840。如果软件确定三角形实际上是透明的,则返回的遍历堆栈包含继续遍历所需的状态。

[0326] (3)在一些实施例中,光束不支持三角形相交,因此遇到的三角形范围默认返回到SM 1840作为“TriRange”碰撞类型,其包括指向与范围重叠的第一个三角形块的指针,指定范围的参数,以及与叶包围盒交点的 t 值。

[0327] ■项目范围,由索引(从存储在complet中的用户提供的“项目范围基础”导出)和项目计数组成。

[0328] 默认情况下,项目范围作为“ItemRange”碰撞类型返回到SM 1840,其由例如索引、计数和与叶包围盒交点的 t 值组成。

[0329] ■实例节点。

[0330] 在一些实施例中,TTU 700可以通过将光线变换为实例BVH的坐标系来本地处理一级实例化。可以用软件处理附加层的实例化(或每个其他层的实例化,取决于策略)。为此提供“实例节点”碰撞类型,包括指向实例节点的指针和与叶包围盒的交点的 t 值。在其他实现中,硬件可以处理两层、三层或更多层的实例化。

[0331] 除了节点特定的碰撞类型之外,还提供了通用的“NodeRef”碰撞类型,其包括指向父complet本身的指针,以及指示与哪个子项相交的ID和该子项与包围盒的交点的 t 值。

[0332] 可以针对不正确地形成查询或BVH或者遍历期间遍历遇到问题的情况提供“错误”碰撞类型。

[0333] 可以为光线或光束未碰撞场景中的所有几何形状的情况提供“无”碰撞类型。

[0334] TTU如何处理四种可能节点类型中的每一种由一组节点特定模式标记确定,该标记被设置为给定光线的查询的一部分。上面提到的“默认”行为对应于模式标记被设置为全零的情况。

[0335] 标记的替代值允许剔除给定类型的所有节点,将给定类型的节点作为NodeRef碰撞类型返回到SM,或者使用它们相应的碰撞类型将三角形范围或实例节点返回到SM,而不

是在TTU 700内本地处理它们。

[0336] 可以提供附加模式标记以用于控制 α 三角形的处理。

[0337] 当配置用于通用并行计算时,与图形处理相比,可以使用更简单的配置。具体地,图17中所示的固定功能图形处理单元被绕过,创建一个更简单的编程模型。在通用并行计算配置中,工作分配单元1725将线程块直接分配和分发给DPC 1820。块中的线程执行相同的程序,在计算中使用唯一的线程ID以确保每个线程生成唯一的结果,使用SM 1840执行程序并执行计算,使用共享存储器/L1高速缓存1970以在线程之间通信,并且使用LSU 1954以通过共享存储器/L1高速缓存1970和存储器分区单元1780读取和写入全局存储器。当配置用于通用并行计算时,SM 1840还可以写入调度器单元1720可以用来在DPC 1820上启动新工作的命令。TTU 700可以用于在通用计算的上下文中加速空间查询。

[0338] PPU 1700可以包括在台式计算机、膝上型计算机、平板计算机、服务器、超级计算机、智能电话(例如,无线、手持设备)、个人数字助理(PDA)、数码相机、车辆、头戴式显示器、手持电子设备等。在一个实施例中,PPU 1700体现在单个半导体衬底上。在另一个实施例中,PPU 1700与一个或多个其他设备(例如附加PPU 1700、存储器1704、精简指令集计算机(RISC)CPU、存储器管理单元(MMU)、数模变换器(DAC)等)一起被包括在片上系统(SoC)中。

[0339] 在一个实施例中,PPU 1700可以被包括在包括一个或多个存储器设备1704的图形卡上。图形卡可以被配置为与台式计算机的主板上的PCIe插槽接口。在又一个实施例中,PPU 1700可以是包括在主板的芯片组中的集成图形处理单元(iGPU)或并行处理器。

[0340] 示例性计算系统

[0341] 具有多个GPU和CPU的系统被用于各种行业,因为开发者在应用(诸如人工智能计算)中暴露和利用更多的并行性。在数据中心、研究机构 and 超级计算机中部署了具有数十至数千个计算节点的高性能GPU加速系统,以解决更大的问题。随着高性能系统内处理设备数量的增加,通信和数据传输机制需要扩展以支持处理设备之间的增加的数据传输。

[0342] 图22是根据一个实施例的使用图17的PPU 1700实现的处理系统1900的概念图。示例性系统1900可以被配置为实现本申请中公开的一种或更多种方法。处理系统1900包括CPU 1930、交换机1910和多个PPU 1700中的每一个以及相应的存储器1704。NVLink 1710提供每个PPU 1700之间的高速通信链路。尽管图22中示出了特定数量的NVLink 1710和互连1702连接,但是到每个PPU 1700和CPU 1930的连接的数量可以改变。交换机1910在互连1702和CPU 1930之间接口。PPU 1700、存储器1704和NVLink 1710可以位于单个半导体平台上以形成并行处理模块1925。在一个实施例中,交换机1912支持在各种不同连接和/或链路之间接口的两个或更多个协议。

[0343] 在另一个实施例(未示出)中,NVLink 1710在每个PPU 1700和CPU 1930之间提供一个或多个高速通信链路,并且交换机1912在互连1702和每个PPU 1700之间进行接口。PPU 1700、存储器1704和互连1702可以位于单个半导体平台上以形成并行处理模块1925。在又一个实施例(未示出)中,互连1702在每个PPU 1700和CPU 1930之间提供一个或多个通信链路,并且交换机1912使用NVLink 1710在每个PPU 1700之间进行接口,以在PPU 1700之间提供一个或多个高速通信链路。在另一个实施例(未示出)中,NVLink 1710在PPU1700和CPU 1930之间通过交换机1912提供一个或多个高速通信链路。在又一个实施

例(未示出)中,互连1702在每个PPU 1700之间直接地提供一个或更多个通信链路。可以使用与NVLink 1710相同的协议将一个或更多个NVLink 1710高速通信链路实现为物理NVLink互连或者片上或裸晶上互连。

[0344] 在本说明书的上下文中,单个半导体平台可以指在裸晶或芯片上制造的唯一的单一的基于半导体的集成电路。应该注意的是,术语单个半导体平台也可以指具有增加的连接的多芯片模块,其模拟片上操作并通过利用常规总线实现方式进行实质性改进。当然,根据用户的需要,各种电路或器件还可以分开放置或以半导体平台的各种组合来放置。可选地,并行处理模块1925可以被实现为电路板衬底,并且PPU 1700和/或存储器1704中的每一个可以是封装器件。在一个实施例中,CPU 1930、交换机1912和并行处理模块1925位于单个半导体平台上。

[0345] 在一个实施例中,每个NVLink 1710的信令速率是20到25千兆位/秒,并且每个PPU 1700包括六个NVLink 1710接口(如图22所示,每个PPU 1700包括五个NVLink 1710接口)。每个NVLink 1710在每个方向上提供25千兆位/秒的数据传输速率,其中六条链路提供1700千兆位/秒的数据传输速率。当CPU 1930还包括一个或更多个NVLink 1710接口时,NVLink 1710可专门用于如图22所示的PPU到PPU通信,或者PPU到PPU以及PPU到CPU的某种组合。

[0346] 在一个实施例中,NVLink 1710允许从CPU 1930到每个PPU 1700的存储器1704的直接加载/存储/原子访问。在一个实施例中,NVLink 1710支持一致性操作,允许从存储器1704读取的数据被存储在CPU 1930的高速缓存层次结构中,减少了CPU 1930的高速缓存访问延迟。在一个实施例中,NVLink 1710包括对地址转换服务(ATS)的支持,允许PPU 1700直接访问CPU 1930内的页表。一个或更多个NVLink 1710还可以被配置为以低功率模式操作。

[0347] 图23示出了示例性系统1965,其中可以实现各种先前实施例的各种体系架构和/或功能。示例性系统1965可以被配置为实现本申请中公开的一种或更多种方法。

[0348] 如图所示,提供系统1965,其包括连接到通信总线1975的至少一个中央处理单元1930。通信总线1975可以使用任何合适的协议来实现,诸如PCI(外围组件互连)、PCI-Express、AGP(加速图形端口)、超传输或任何其他总线或一个或更多个点通信协议。系统1965还包括主存储器1940。控制逻辑(软件)和数据被存储在主存储器1940中,主存储器1940可以采取随机存取存储器(RAM)的形式。

[0349] 系统1965还包括输入设备1960、并行处理系统1925和显示设备1945,即常规CRT(阴极射线管)、LCD(液晶显示器)、LED(发光二极管)、等离子显示器等。可以从输入设备1960(例如键盘、鼠标、触摸板、麦克风等)接收用户输入。前述模块和/或设备中的每一个甚至可以位于单个半导体平台上以形成系统1965。可选地,根据用户的需要,各个模块还可以分开放置或以半导体平台的各种组合来放置。

[0350] 此外,系统1965可以出于通信目的通过网络接口1935耦合到网络(例如,电信网络、局域网(LAN)、无线网络、广域网(WAN)(诸如因特网)、对等网络、电缆网络等)。

[0351] 系统1965还可以包括辅助存储(未示出)。辅助存储610包括例如硬盘驱动器和/或可移除存储驱动器、代表软盘驱动器、磁带驱动器、光盘驱动器、数字多功能盘(DVD)驱动器、记录设备、通用串行总线(USB)闪存。可移除存储驱动器以众所周知的方式从可移除存储单元读取和/或写入可移除存储单元。

[0352] 计算机程序或计算机控制逻辑算法可以存储在主存储器1940和/或辅助存储中。

这些计算机程序在被执行时使得系统1965能够执行各种功能。存储器1940、存储和/或任何其他存储是计算机可读介质的可能示例。

[0353] 各种在先附图的架构和/或功能可以在通用计算机系统、电路板系统、专用于娱乐目的的游戏控制台系统、专用系统和/或任何其他所需的系统的上下文中实现。例如,系统1965可以采取台式计算机、膝上型计算机、平板电脑、服务器、超级计算机、智能电话(例如,无线、手持设备)、个人数字助理(PDA)、数字相机、运载工具、头戴式显示器、手持式电子设备、移动电话设备、电视机、工作站、游戏控制台、嵌入式系统和/或任何其他类型的逻辑的形式。

[0354] 虽然上面已经描述了各种实施例,但是应该理解,它们仅以示例的方式呈现而不是限制。因此,优选实施例的宽度和范围不应受任何上述示例性实施例的限制,而应仅根据所附权利要求及其等同物来限定。

[0355] 图形处理管线

[0356] 在一个实施例中,PPU 1700包括图形处理单元(GPU)。PPU 1700被配置为接收指定用于处理图形数据的着色器程序的命令。图形数据可以被定义为一组图元,例如点、线、三角形、四边形、三角形条等。通常,图元包括指定图元的多个顶点的数据(例如,在模型空间坐标系中)以及与图元的每个顶点相关联的属性。PPU 1700可以被配置为处理图形图元以生成帧缓冲器(即,用于显示器的每个像素的像素数据)。

[0357] 应用程序将场景的模型数据(即,顶点和属性的集合)写入存储器,诸如系统存储器或存储器1704。模型数据定义可在显示器上可见的每个对象。然后,应用程序对驱动程序内核进行API调用,以请求呈现和显示模型数据。驱动程序内核读取模型数据并将命令写入一个或更多个流以执行处理模型数据的操作。命令可以引用要在PPU 1700的SM 1840上实现的不同着色器程序,包括顶点着色器、外壳着色器、域着色器、几何着色器、像素着色器、光线生成着色器、光线相交着色器、光线碰撞着色器和光线错过着色器中的一个或更多个(这些着色器对应于DXR API定义的着色器,忽略“最近碰撞”和“任意碰撞”着色器之间的任何区别;请参阅<https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-ray-tracing/>)。例如,SM 1840中的一个或更多个可以被配置为执行顶点着色器程序,该顶点着色器程序处理由模型数据定义的多个顶点。在一个实施例中,不同的SM 1840可以被配置为同时执行不同的着色器程序。例如,SM 1840的第一子集可以被配置为执行顶点着色器程序,而SM 1840的第二子集可以被配置为执行像素着色器程序。SM 1840的第一子集处理顶点数据以产生经处理的顶点数据,并将经处理的顶点数据写入L2高速缓存1860和/或存储器1704。在经处理的顶点数据被光栅化(即,从三维数据变换成屏幕空间中的二维数据)以产生片段数据之后,SM 1840的第二子集执行像素着色器以产生经处理的片段数据,然后将其与其他经处理的片段数据混合并写入存储器1704中的帧缓冲器。顶点着色器程序和像素着色器程序可以同时执行,以管线方式处理来自同一场景的不同数据,直到场景的所有模型数据都已被渲染到帧缓冲器。然后,帧缓冲器的内容被发送到显示控制器以在显示设备上显示。

[0358] 图24是由图17的PPU 1700实现的图形处理管线2000的概念图。图形处理管线2000是被实现为从3D几何数据生成2D计算机生成的图像的处理步骤的抽象流程图。众所周知,管线架构可以通过将操作分成多个阶段来更有效地执行长延迟操作,其中每个阶段的输出

耦合到下一个连续阶段的输入。因此,图形处理管线2000接收从图形处理管线2000的一个阶段发送到下一个阶段的输入数据2001以生成输出数据2002。在一个实施例中,图形处理管线2000可以表示由 OpenGL®API定义的图形处理管线。作为选择,图形处理管线2000可以在前面的附图和/或任何后续附图的功能和架构的上下文中实现。

[0359] 如图24所示,图形处理管线2000包括包括多个阶段的管线架构。这些阶段包括但不限于数据组装阶段2010,顶点着色阶段2020,图元组装阶段2030,几何着色阶段2040,视口缩放、剔除和裁剪(VSCC)阶段2050,光栅化阶段2060,片段着色阶段2070和光栅操作阶段2080。在一个实施例中,输入数据2001包括配置处理单元以实现图形处理管线2000的阶段的命令和由各阶段处理的几何图元(例如,点、线、三角形、四边形、三角形条或扇形等)。输出数据2002可以包括被复制到存储器中的帧缓冲器或其他类型的表面数据结构的像素数据(即,颜色数据)。

[0360] 数据组装阶段2010接收输入数据2001,其指定高阶表面的顶点数据、图元等。数据组装阶段2010在临时存储器或队列中收集顶点数据,例如通过从主机处理器接收包括指向存储器中的缓冲器的指针并从缓冲器读取顶点数据的命令。然后将顶点数据发送到顶点着色阶段2020以进行处理。

[0361] 顶点着色阶段2020通过对每个顶点执行一次一组操作(即,顶点着色器或程序)来处理顶点数据。例如,顶点可以例如被指定为与一个或多个顶点属性(例如,颜色、纹理坐标、表面法线等)相关联的4坐标向量(即, $\langle x, y, z, w \rangle$)。顶点着色阶段2020可以操纵各个顶点属性,例如位置、颜色、纹理坐标等。换句话说,顶点着色阶段2020对顶点坐标或与顶点相关联的其他顶点属性执行操作。这些操作通常包括照明操作(即,修改顶点的颜色属性)和变换操作(即,修改顶点的坐标空间)。例如,可以使用对象坐标空间中的坐标来指定顶点,通过将坐标乘以将坐标从对象坐标空间变换为世界空间或归一化设备坐标(NCD)空间的矩阵来进行变换。顶点着色阶段2020生成变换的顶点数据,该变换的顶点数据被发送到图元组装阶段2030。

[0362] 图元组装阶段2030收集由顶点着色阶段2020输出的顶点,并将顶点分组为几何图元,以供几何着色阶段2040处理。例如,图元组装阶段2030可被配置为将每三个连续顶点分组为几何图元(即,三角形),用于发送到几何着色阶段2040。在一些实施例中,特定顶点可以重复用于连续几何图元(例如,三角形条带中的两个连续三角形可以共享两个顶点)。图元组装阶段2030将几何图元(即,关联顶点的集合)发送到几何着色阶段2040。

[0363] 几何着色阶段2040通过对几何图元执行一组操作(即,几何着色器或程序)来处理几何图元。曲面细分操作可以从每个几何图元生成一个或多个几何图元。换句话说,几何着色阶段2040可以将每个几何图元细分为两个或多个几何图元的更精细网格,以供图形处理管线2000的其余部分处理。几何着色阶段2040将几何图元发送到视口SCC阶段2050。

[0364] 在一个实施例中,图形处理管线2000可以在流式多处理器内操作,并且顶点着色阶段2020、图元组装阶段2030、几何着色阶段2040、片段着色阶段2070、和/或与其相关联的硬件/软件可以顺序地执行处理操作。一旦顺序处理操作完成,在一个实施例中,视口SCC阶段2050可以利用该数据。在一个实施例中,可以将由图形处理管线2000中的一个或多个阶段处理的图元数据写入高速缓存(例如,L1高速缓存、顶点高速缓存等)。在这种情况下,在一个实施例中,视口SCC阶段2050可以访问高速缓存中的数据。在一个实施例中,视口SCC

阶段2050和光栅化阶段2060被实现为固定功能电路。

[0365] 视口SCC阶段2050执行几何图元的视口缩放、剔除和裁剪。渲染的每个表面与抽象摄像机位置相关联。摄像机位置表示观看者观看场景的位置,并定义包围场景对象的视锥体。视锥体可包括观察平面,后平面和四个裁剪平面。完全在视锥体之外的任何几何图元可以被剔除(即,丢弃),因为该几何图元将不会对最终渲染场景做出贡献。可以裁剪部分位于视锥体内部并且部分位于视锥体外部的任何几何图元(即,变换为包围在视锥体内的新的几何图元)。此外,几何图元可以各自基于视锥体的深度来缩放。然后,将所有可能可见的几何图元发送到光栅化阶段2060。

[0366] 光栅化阶段2060将3D几何图元转换为2D片段(例如,能够用于显示等)。光栅化阶段2060可以被配置为利用几何图元的顶点来设置一组平面方程,从中可以插入各种属性。光栅化阶段2060还可以计算多个像素的覆盖掩码,其指示像素的一个或更多个样本位置是否拦截几何图元。在一个实施例中,还可以执行z测试以确定几何图元是否被已经被光栅化的其他几何图元遮挡。光栅化阶段2060生成片段数据(即,与每个被覆盖像素的特定样本位置相关联的内插顶点属性),其被发送到片段着色阶段2070。

[0367] 片段着色阶段2070通过对每个片段执行一组操作(即,片段着色器或程序)来处理片段数据。片段着色阶段2070可以生成片段的像素数据(即,颜色值),诸如通过使用片段的内插纹理坐标执行照明操作或采样纹理映射。片段着色阶段2070生成发送到光栅操作阶段2080的像素数据。

[0368] 光栅操作阶段2080可以对像素数据执行各种操作,诸如执行阿尔法测试,模板测试,以及将像素数据与对应于与像素相关联的其他片段的其他像素数据混合。当光栅操作阶段2080已经完成处理像素数据(即,输出数据2002)时,可以将像素数据写入渲染目标(诸如帧缓冲器、颜色缓冲器等)。

[0369] 应当理解,除了上述一个或更多个阶段之外或代替上述一个或更多个阶段,可以在图形处理管线2000中包括一个或更多个附加阶段。抽象图形处理管线的各种实现可以实现不同的阶段。此外,在一些实施例中(例如,几何着色阶段2040),可以从图形处理管线中排除上述一个或更多个阶段。其他类型的图形处理管线也考虑在本公开的范围。此外,图形处理管线2000的任何阶段可以由图形处理器(诸如PPU 200)内的一个或更多个专用硬件单元实现。图形处理管线2000的其他阶段可以由可编程硬件单元(诸如PPU 1700的SM 1840的)实现。

[0370] 图形处理管线2000可以经由由主机处理器(诸如CPU 120)执行的应用来实现。在一个实施例中,设备驱动器可以实现定义可以由应用程序利用以生成用于显示的图形数据的各种功能的应用程序编程接口(API)。设备驱动程序是包括控制PPU 1700的操作的多个指令的软件程序。API为程序员提供抽象,其允许程序员利用专用图形硬件(例如PPU 1700)来生成图形数据,而不需要程序员利用PPU 1700的特定指令集。应用程序可以包括被路由到PPU 1700的设备驱动器的API调用。设备驱动器解释API调用并执行各种操作以响应API调用。在某些情况下,设备驱动器可以通过在CPU上执行指令来执行操作。在其他情况下,设备驱动器可以至少部分地通过利用CPU和PPU 1700之间的输入/输出接口在PPU 1700上启动操作来执行操作。在一个实施例中,设备驱动器被配置为利用PPU 1700的硬件来实现图形处理管线2000。

[0371] 可以在PPU 1700内执行各种程序以便实现图形处理管线2000的各个阶段。例如,设备驱动器可以在PPU 1700上启动内核,以在一个SM 1840(或多个SM 1840)上执行顶点着色阶段2020。设备驱动器(或由PPU 1800执行的初始内核)还可以在PPU 1800上启动其他内核,以执行图形处理管线2000的其他阶段,例如几何着色阶段2040和片段着色阶段2070。此外,图形处理管线2000的一些阶段可以在固定单元硬件上实现,例如在PPU 1800内实现的光栅化器或数据汇编器上。可以理解,来自一个内核的结果可以在由SM 1840上的后续内核处理之前通过一个或更多个中间固定功能硬件单元来处理。

[0372] 机器学习

[0373] 在处理器(诸如PPU 1700)上开发的深度神经网络(DNN)已经用于各种使用情况:从自驾车到更快药物开发,从在线图像数据库中的自动图像字幕到视频聊天应用中的智能实时语言翻译。深度学习是一种技术,它建模人类大脑的神经学习过程,不断学习,不断变得更聪明,并且随着时间的推移更快地传送更准确的结果。一个孩子最初是由成人教导,以正确识别和分类各种形状,最终能够在没有任何辅导的情况下识别形状。同样,深度学习或神经学习系统需要在对象识别和分类方面进行训练,以便在识别基本对象、遮挡对象等同时还有为对象分配情景时变得更加智能和高效。

[0374] 在最简单的层面上,人类大脑中的神经元查看接收到的各种输入,将重要性级别分配给这些输入中的每一个,并且将输出传递给其他神经元以进行处理。人造神经元或感知器是神经网络的最基本模型。在一个示例中,感知器可以接收一个或更多个输入,其表示感知器正被训练为识别和分类的对象的各种特征,并且在定义对象形状时,这些特征中的每一个基于该特征的重要性赋予一定的权重。

[0375] 深度神经网络(DNN)模型包括许多连接的感知器(例如节点)的多个层,其可以用大量输入数据来训练以快速高精度地解决复杂问题。在一个示例中,DLL模型的第一层将汽车的输入图像分解为各个部分,并查找基本图案(诸如线条和角)。第二层组装线条以寻找更高层的图案,诸如轮子、挡风玻璃和镜子。下一层识别运载工具类型,最后几层为输入图像生成标签,识别特定汽车品牌的型号。

[0376] 一旦DNN被训练,DNN就可以被部署并用于在被称为推理(inference)的过程中识别和分类对象或图案。推理的示例(DNN从给定输入中提取有用信息的过程)包括识别存入ATM机中的支票上的手写数字、识别照片中朋友的图像、向超过五千万用户提供电影推荐、识别和分类不同类型的汽车、行人和无人驾驶汽车中的道路危险或实时翻译人类言语。

[0377] 在训练期间,数据在前向传播阶段流过DNN,直到产生预测为止,其指示对应于输入的标签。如果神经网络没有正确标记输入,则分析正确标签和预测标签之间的误差,并且在后向传播阶段期间针对每个特征调整权重,直到DNN正确标记该输入和训练数据集中的其他输入为止。训练复杂的神经网络需要大量的并行计算性能,包括由PPU 1700支持的浮点乘法和加法。与训练相比,推理的计算密集程度比训练低,其是一个延迟敏感过程,其中经训练的神经网络应用于它以前没有见过的新的输入,以进行图像分类、翻译语音以及通常推理新的信息。

[0378] 神经网络严重依赖于矩阵数学运算,并且复杂的多层网络需要大量的浮点性能和带宽来提高效率和速度。采用数千个处理核心,针对矩阵数学运算进行了优化,并传送数十到数百TFLOPS的性能,PPU 1700是能够传送基于深度神经网络的人工智能和机器学习应用

所需性能的计算平台。

[0379] 虽然已经结合目前被认为是最实用和优选的实施例描述了本发明,但是应该理解,本发明不限于所公开的实施例,而是相反,旨在涵盖各种实施例。在所附权利要求的精神和范围内包括的修改和等同布置。

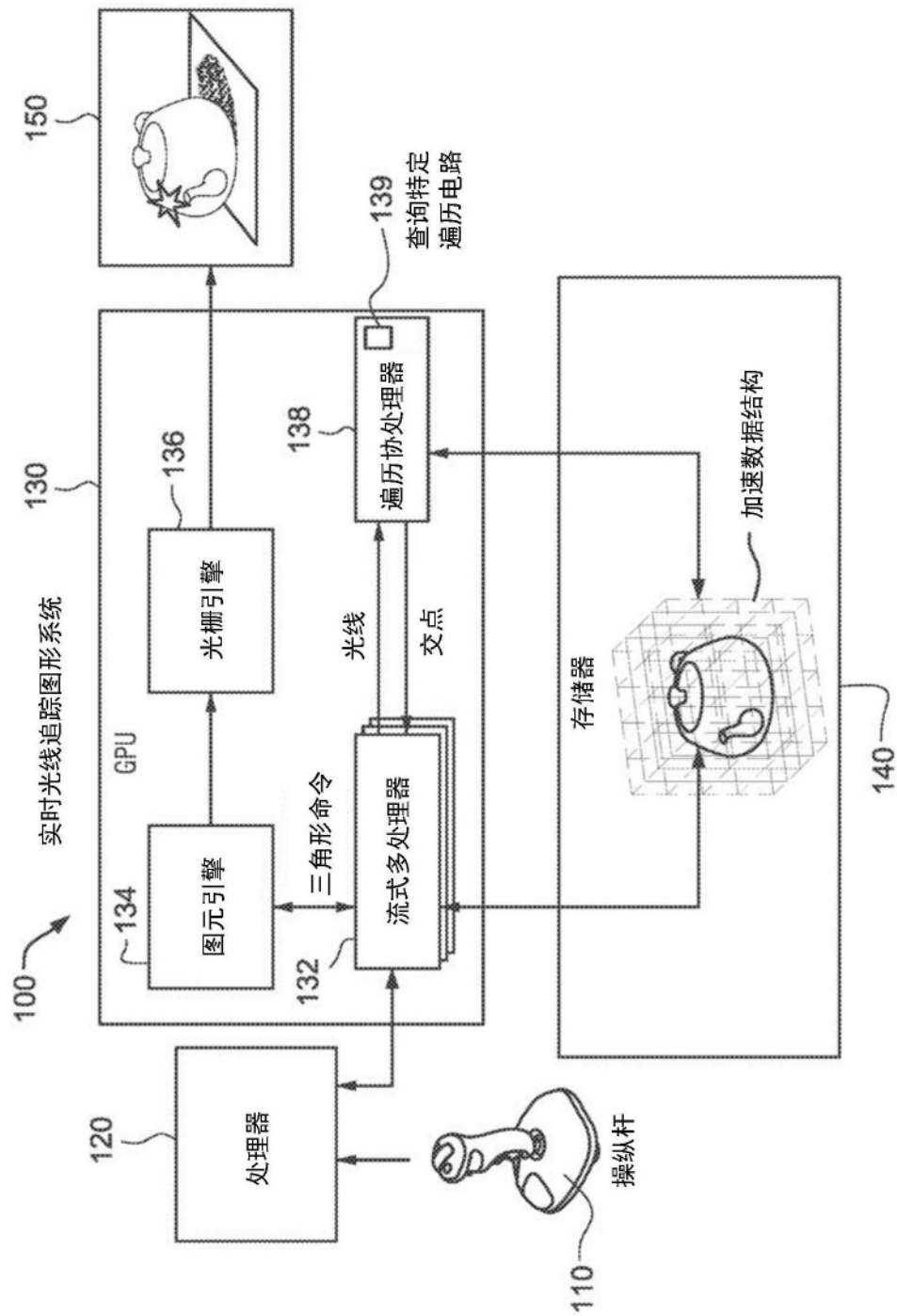


图1



图2A

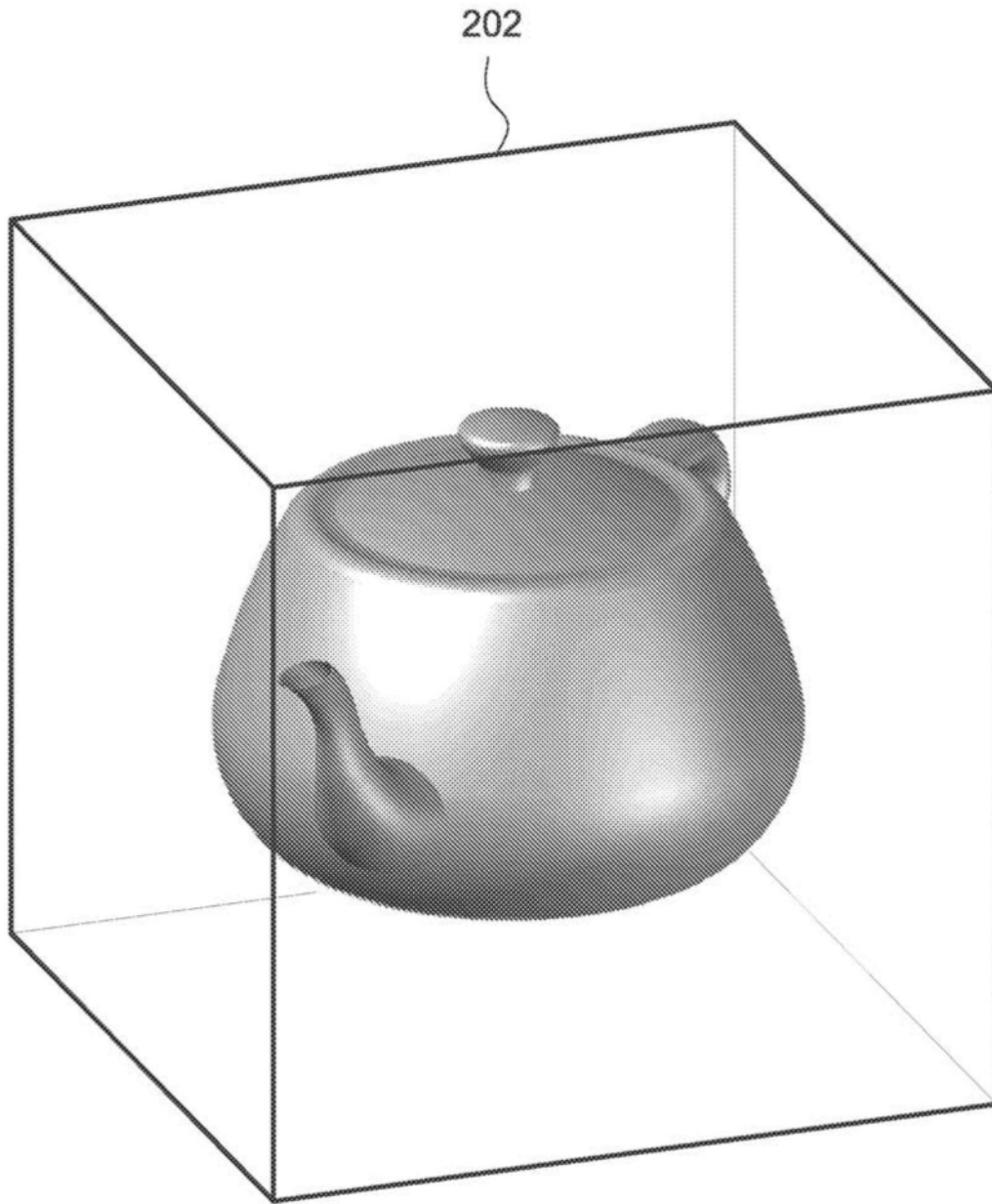


图2B

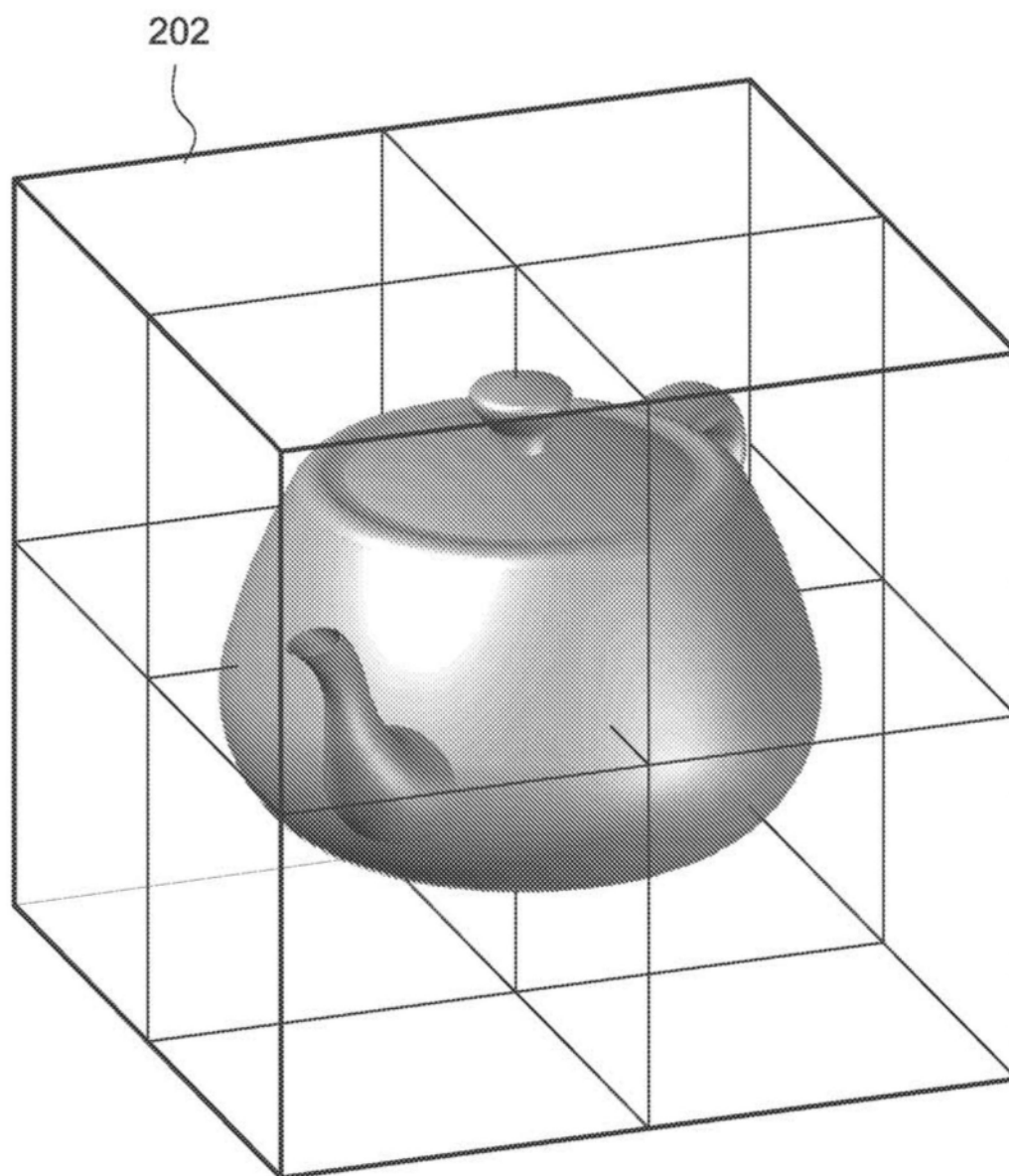


图2C

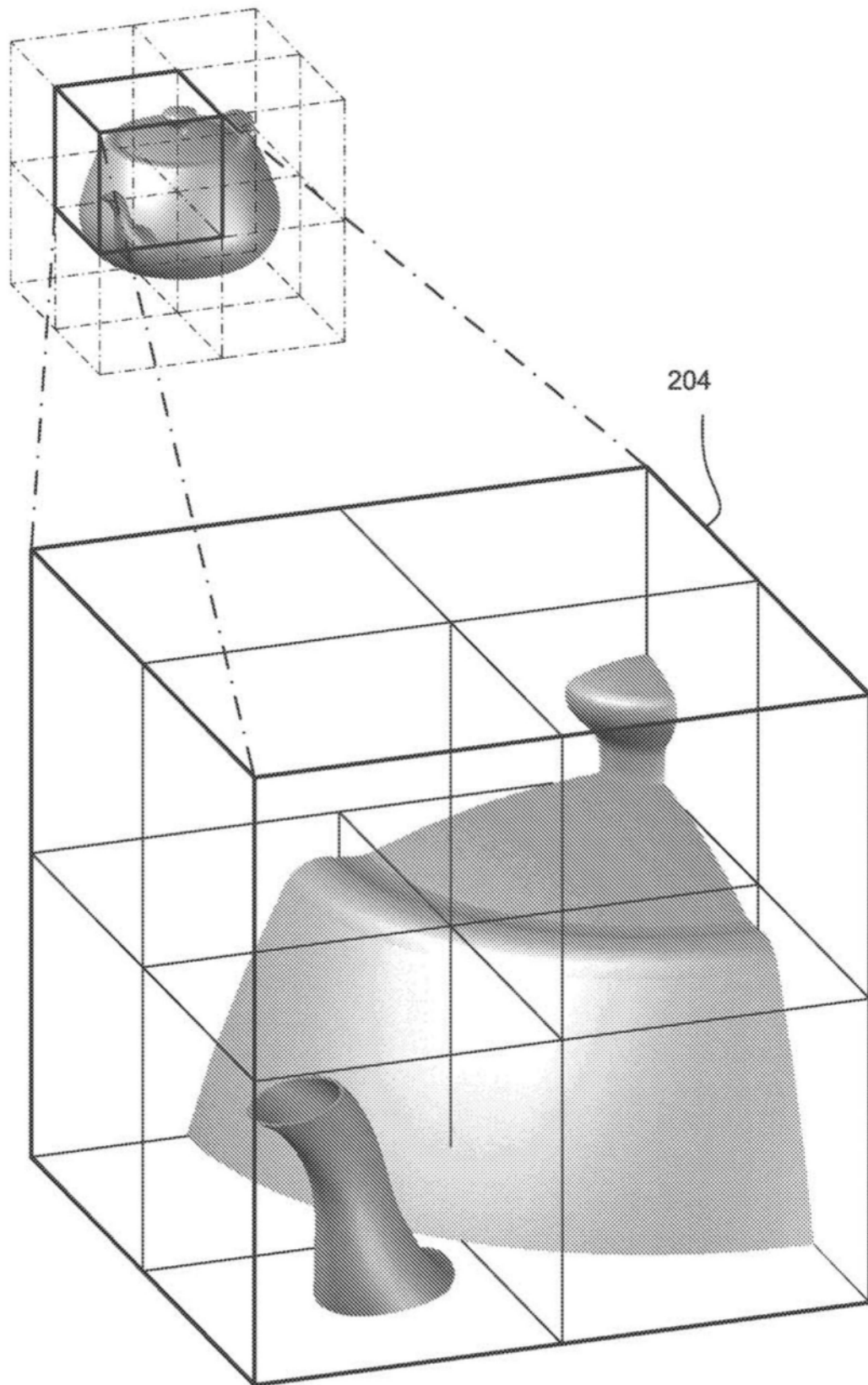


图2D

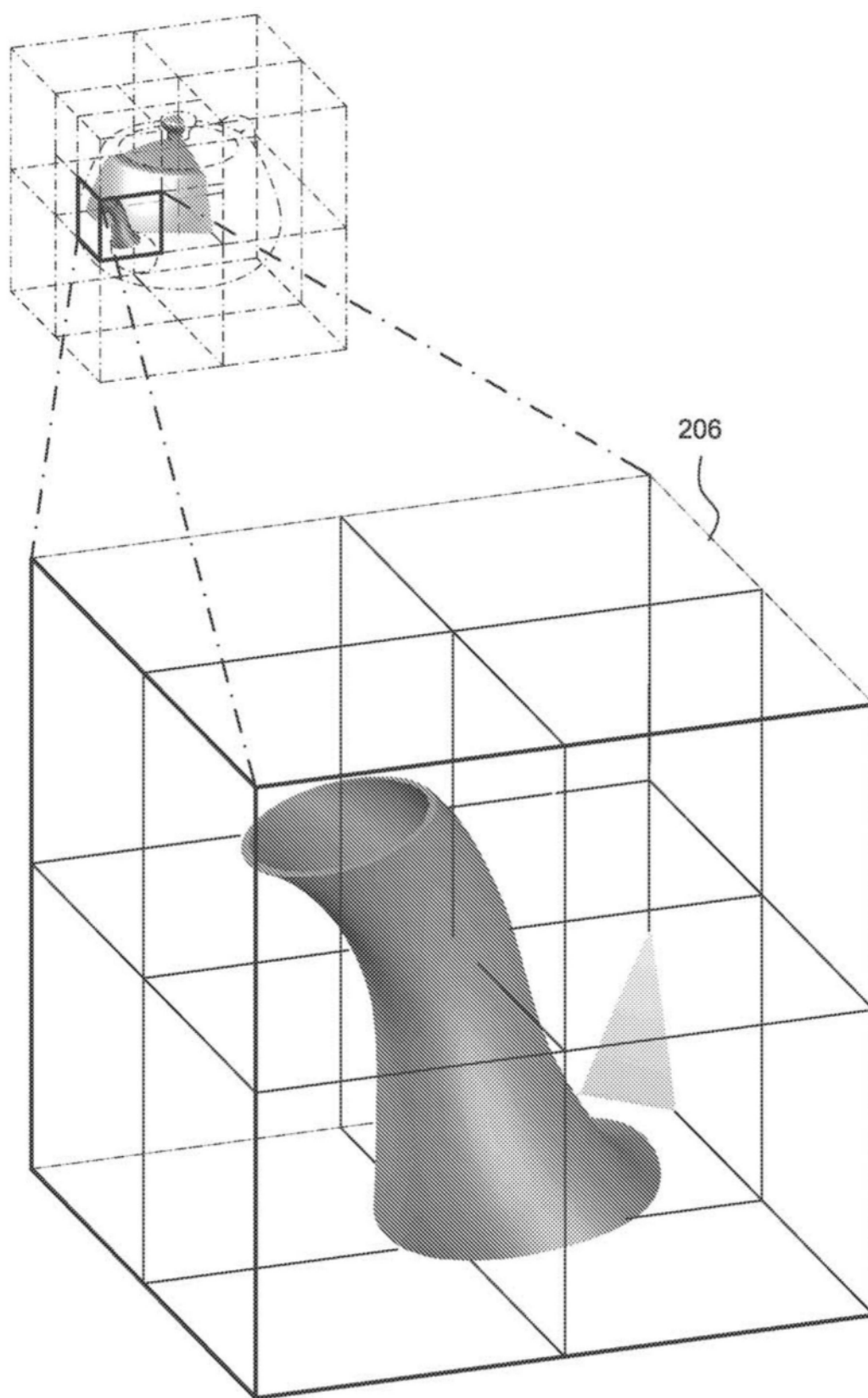


图2E

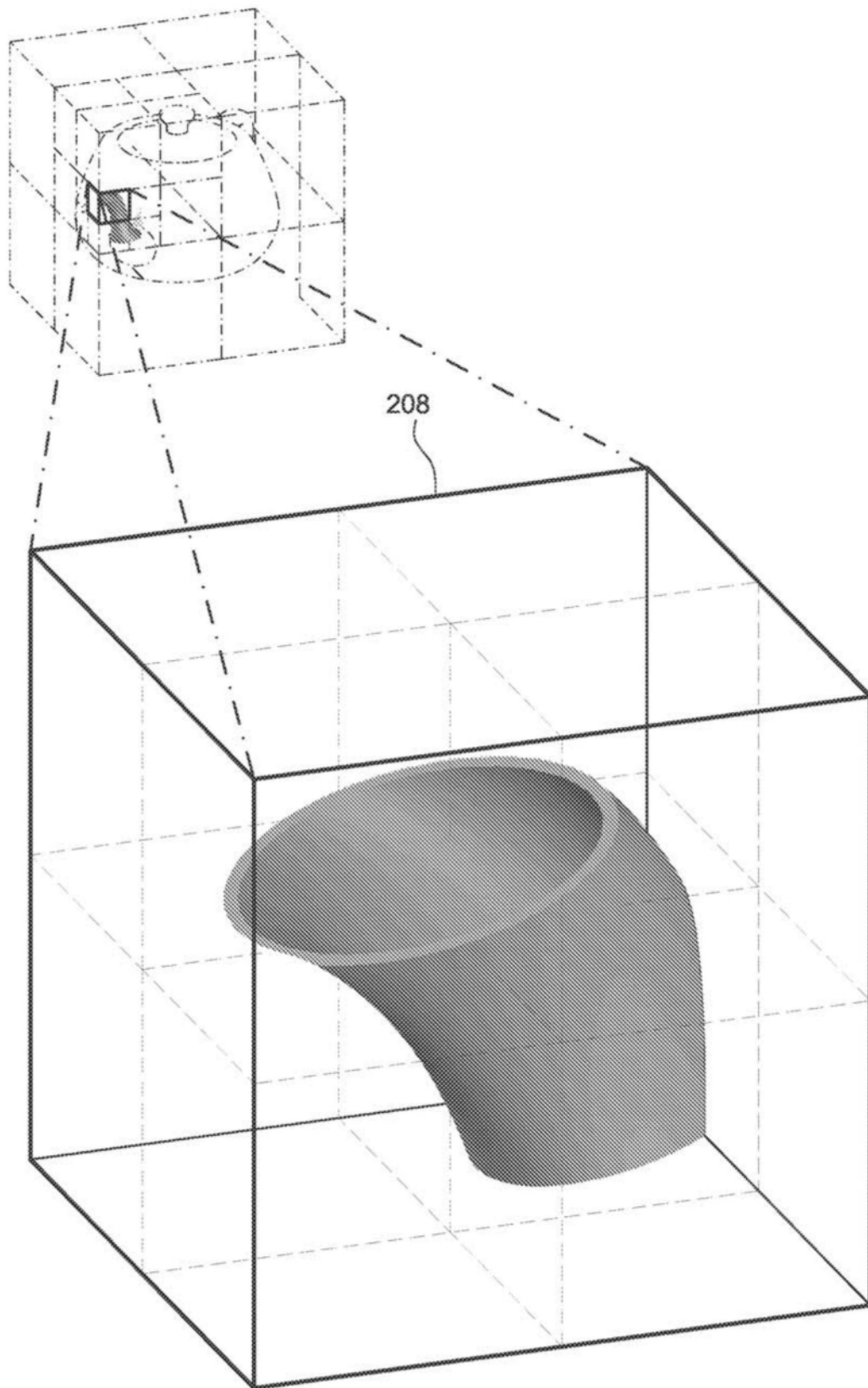


图2F

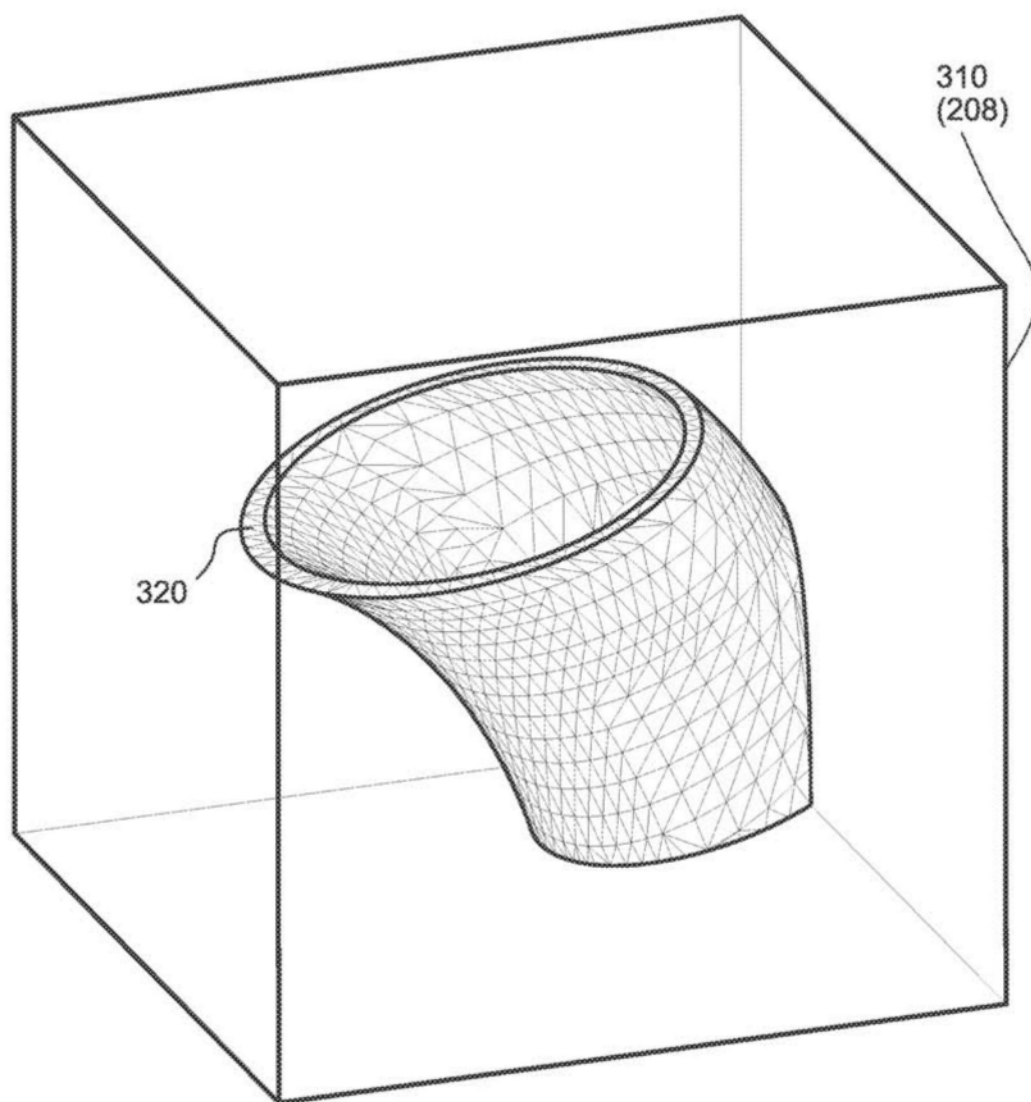


图2G

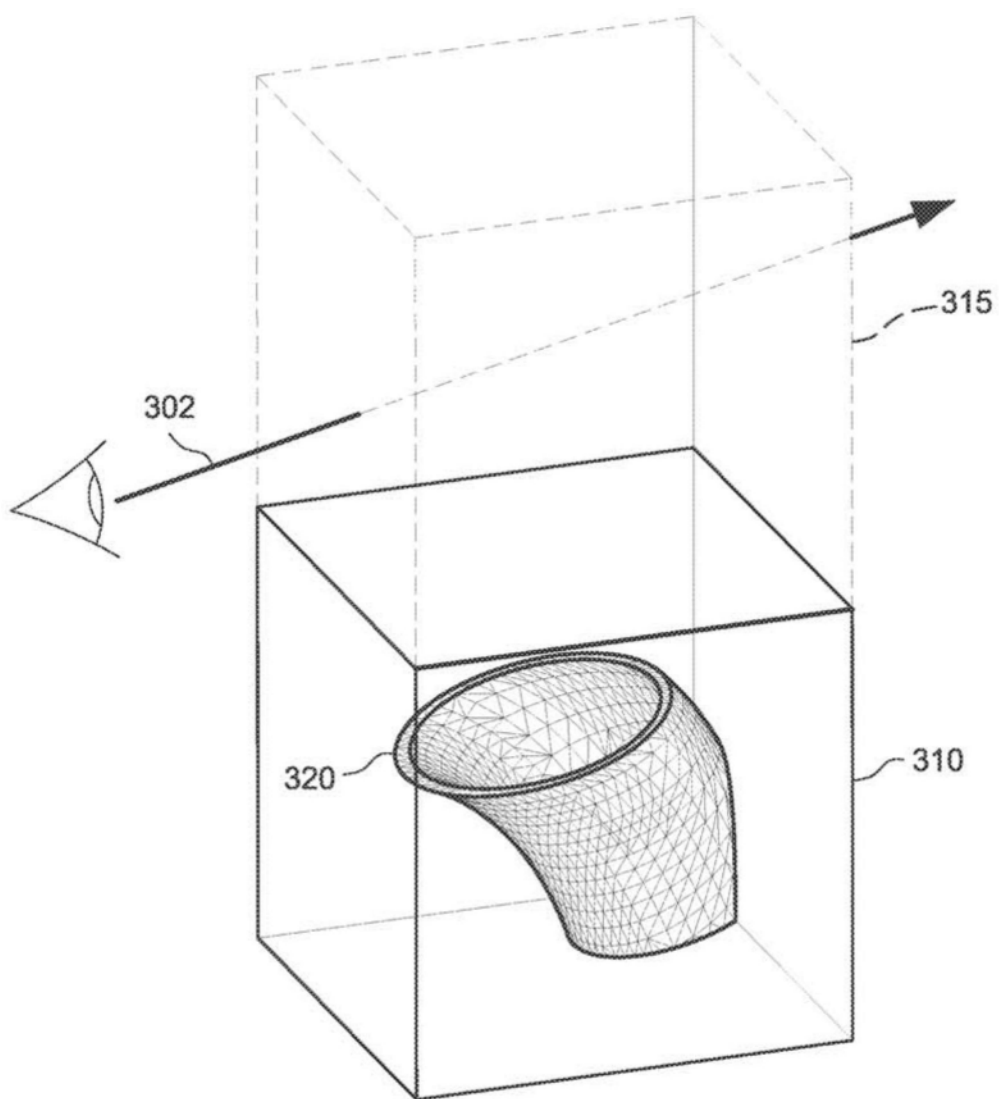


图3A

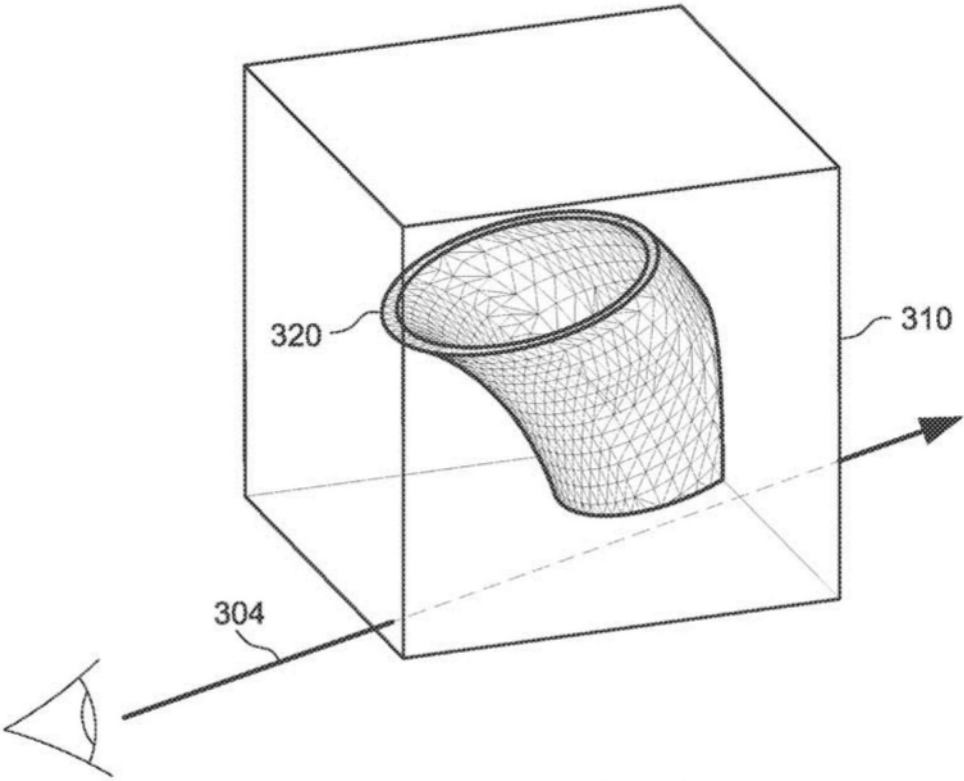


图3B

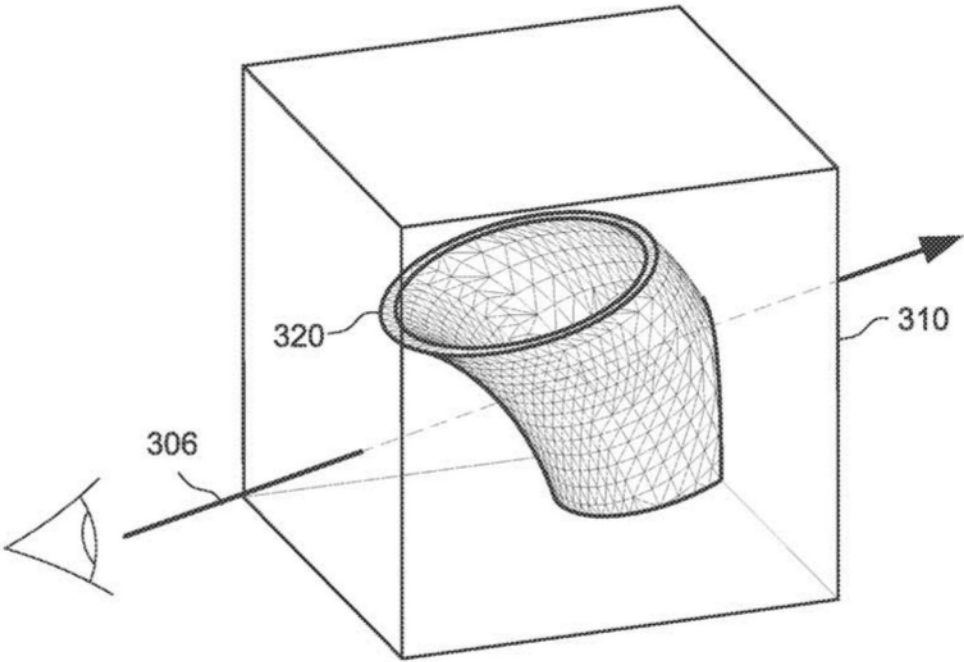


图3C

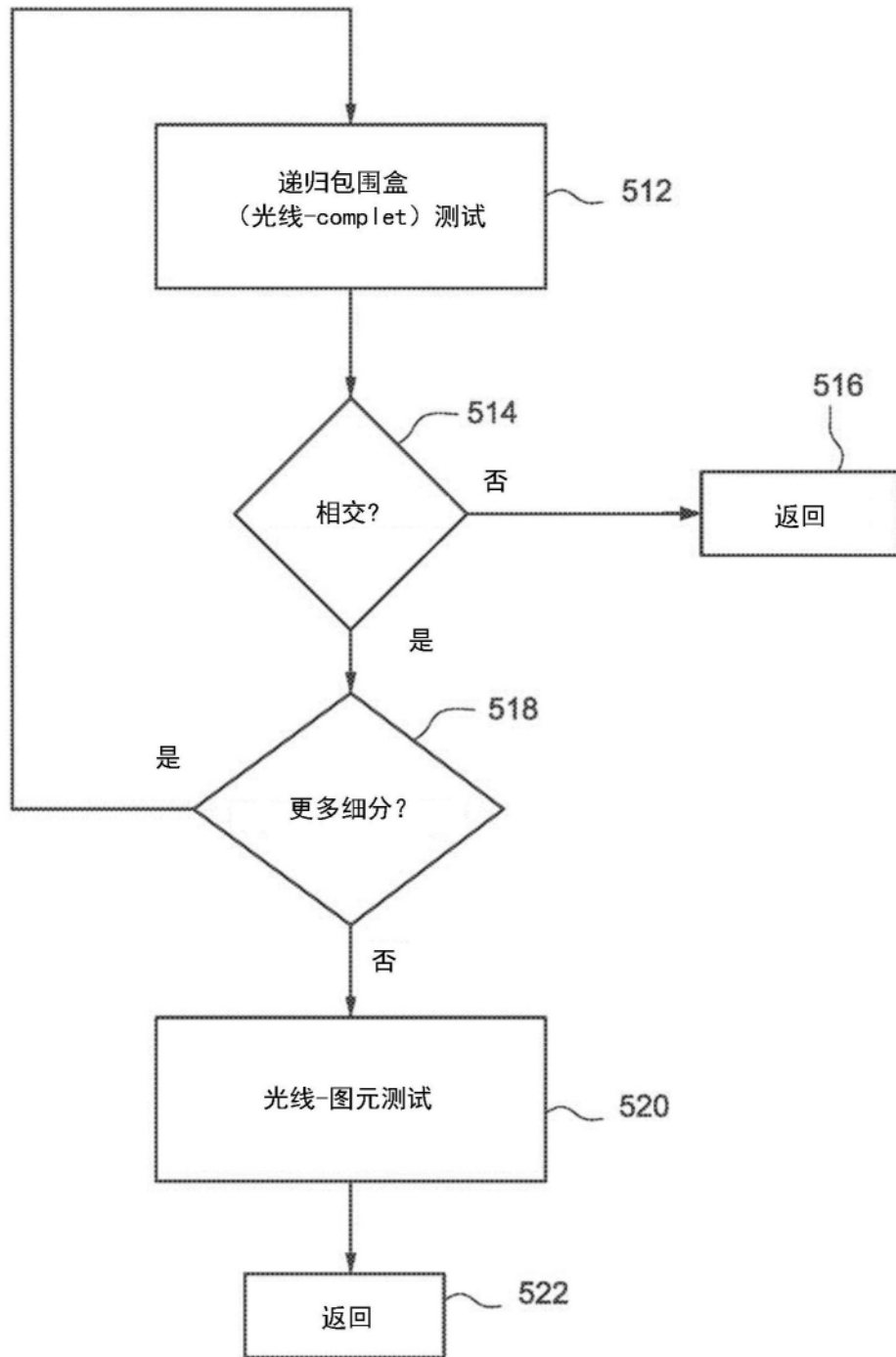


图4

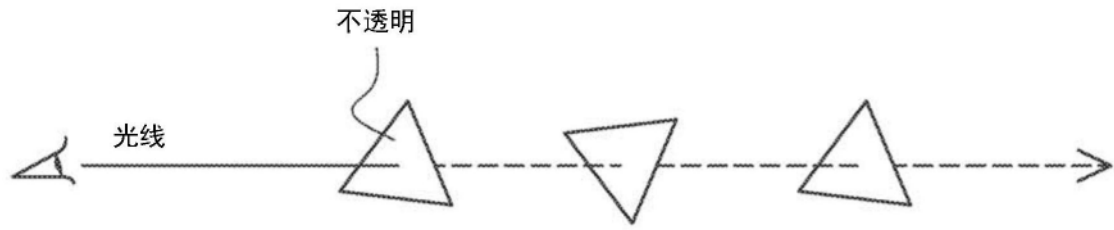


图5A

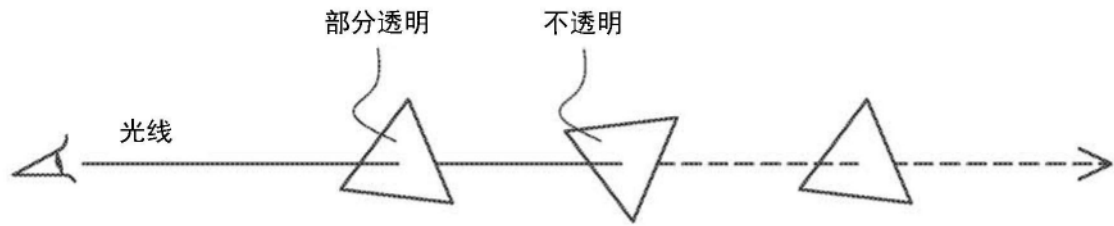


图5B

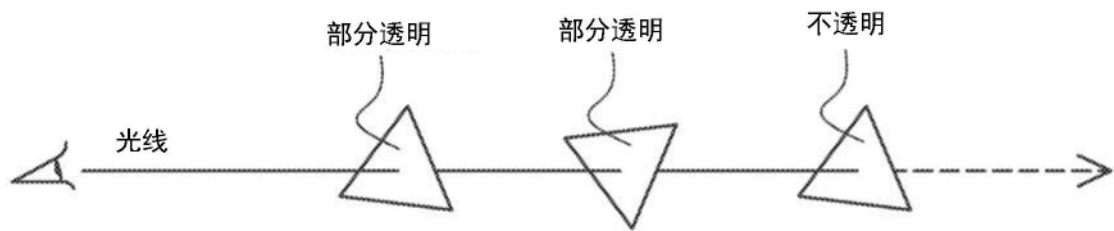


图5C

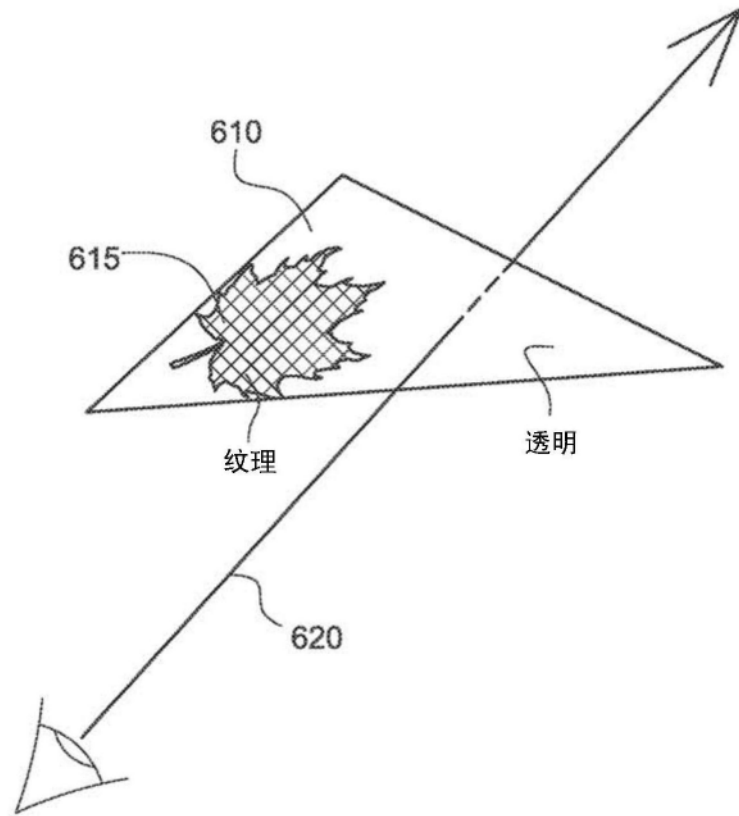


图6A

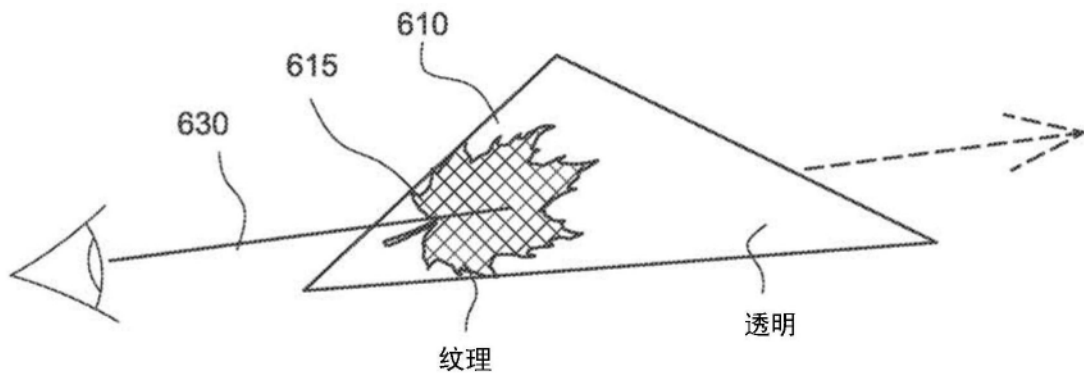


图6B

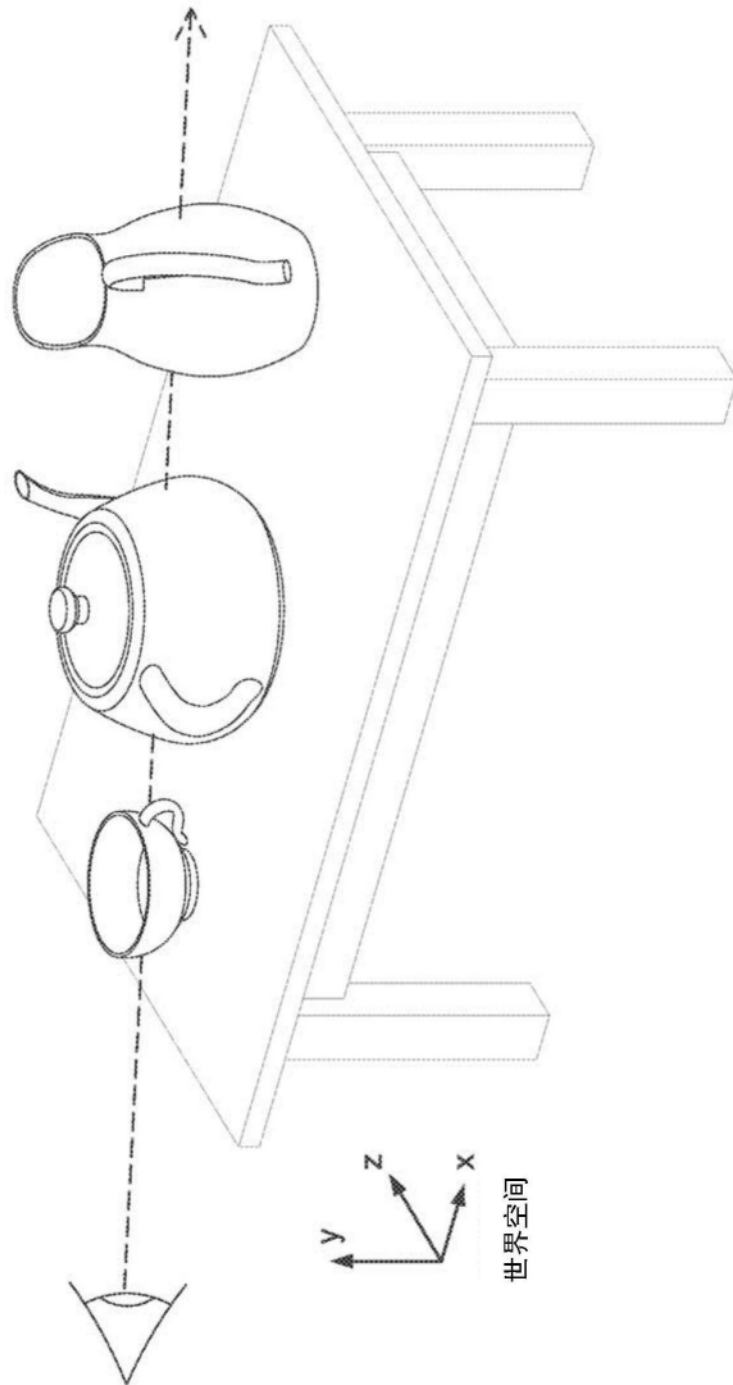


图7A

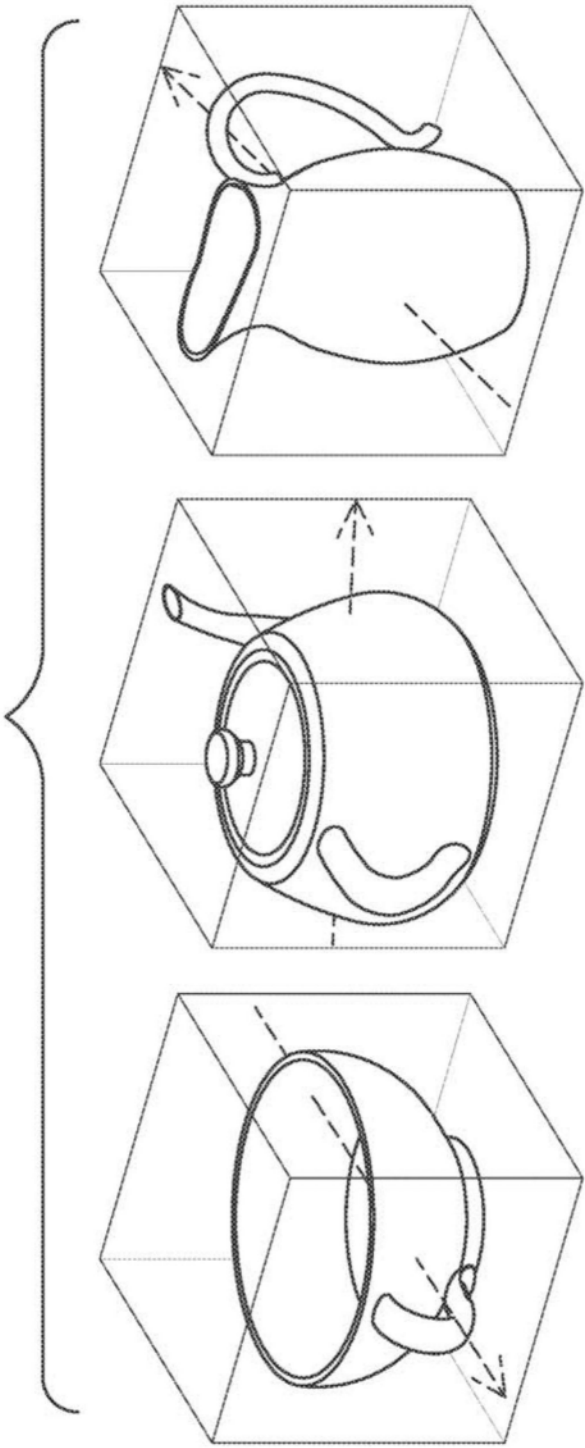


图7B

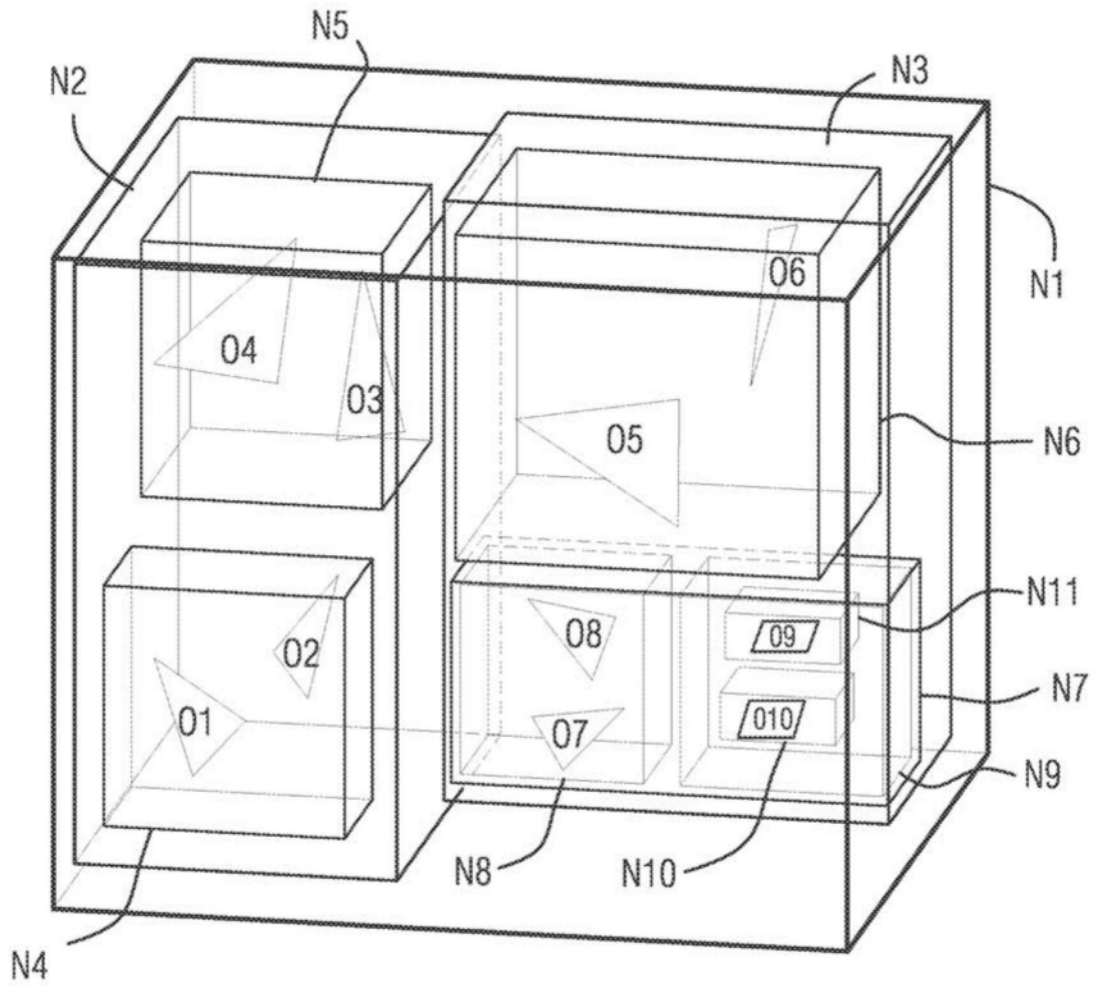


图8A

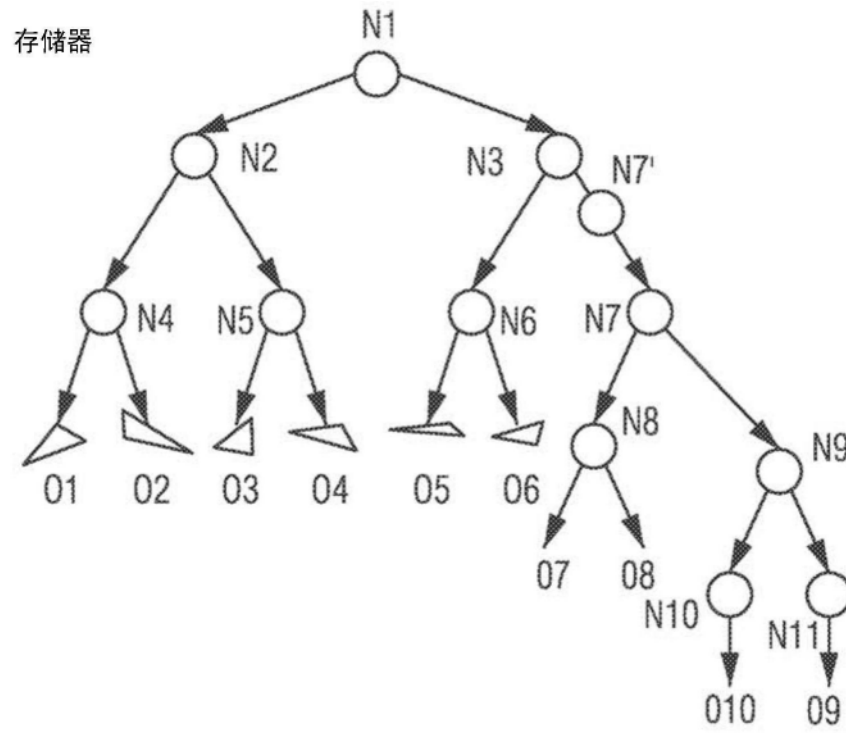


图8B

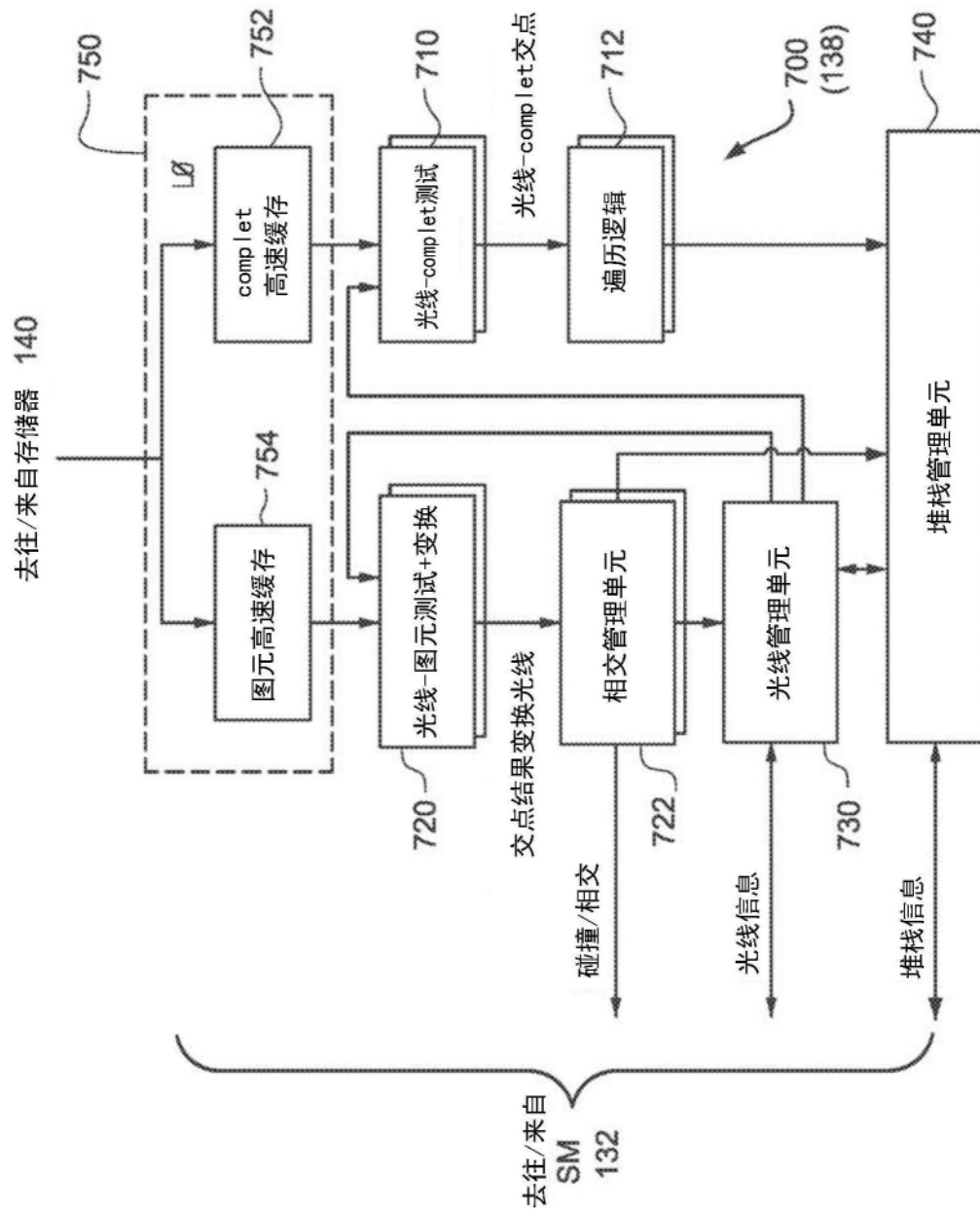


图9

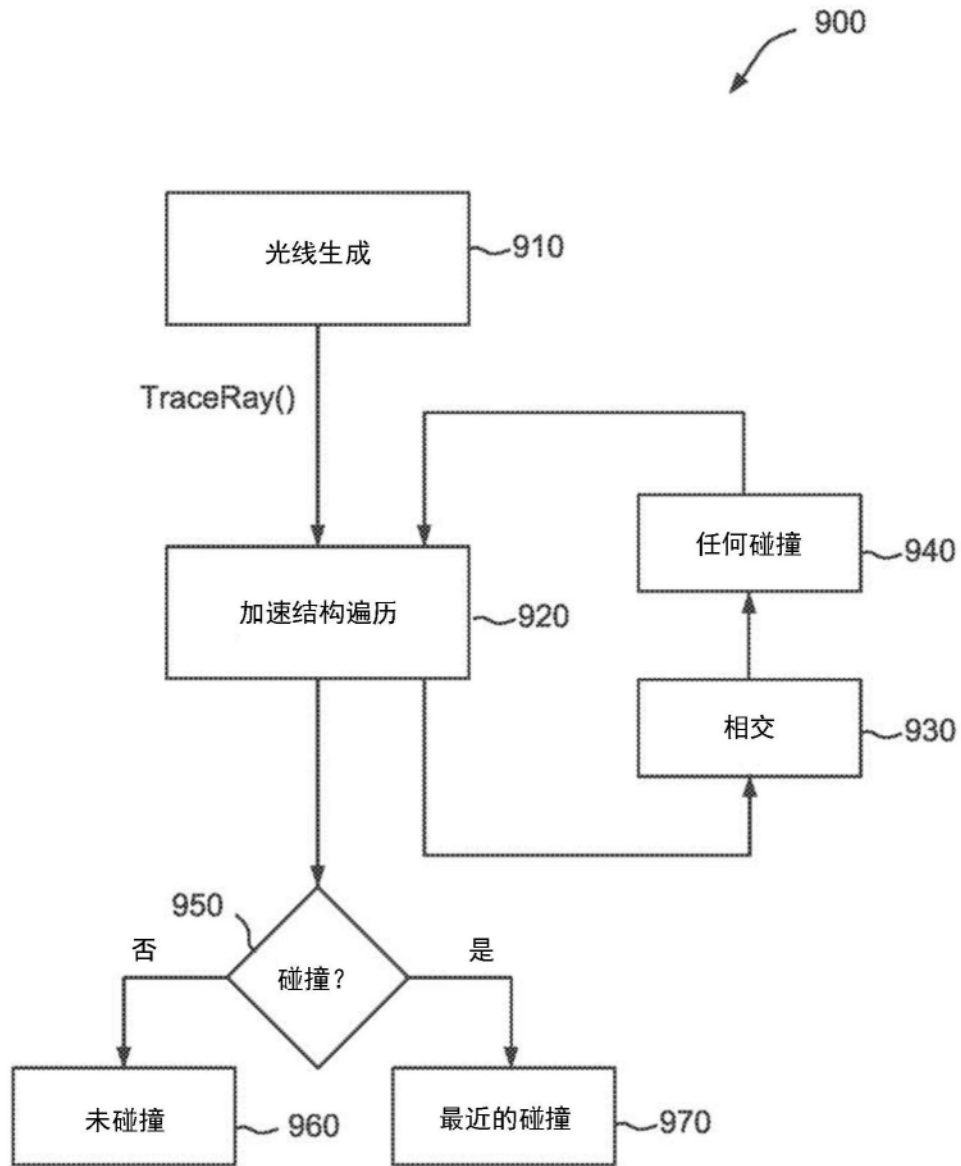


图10A

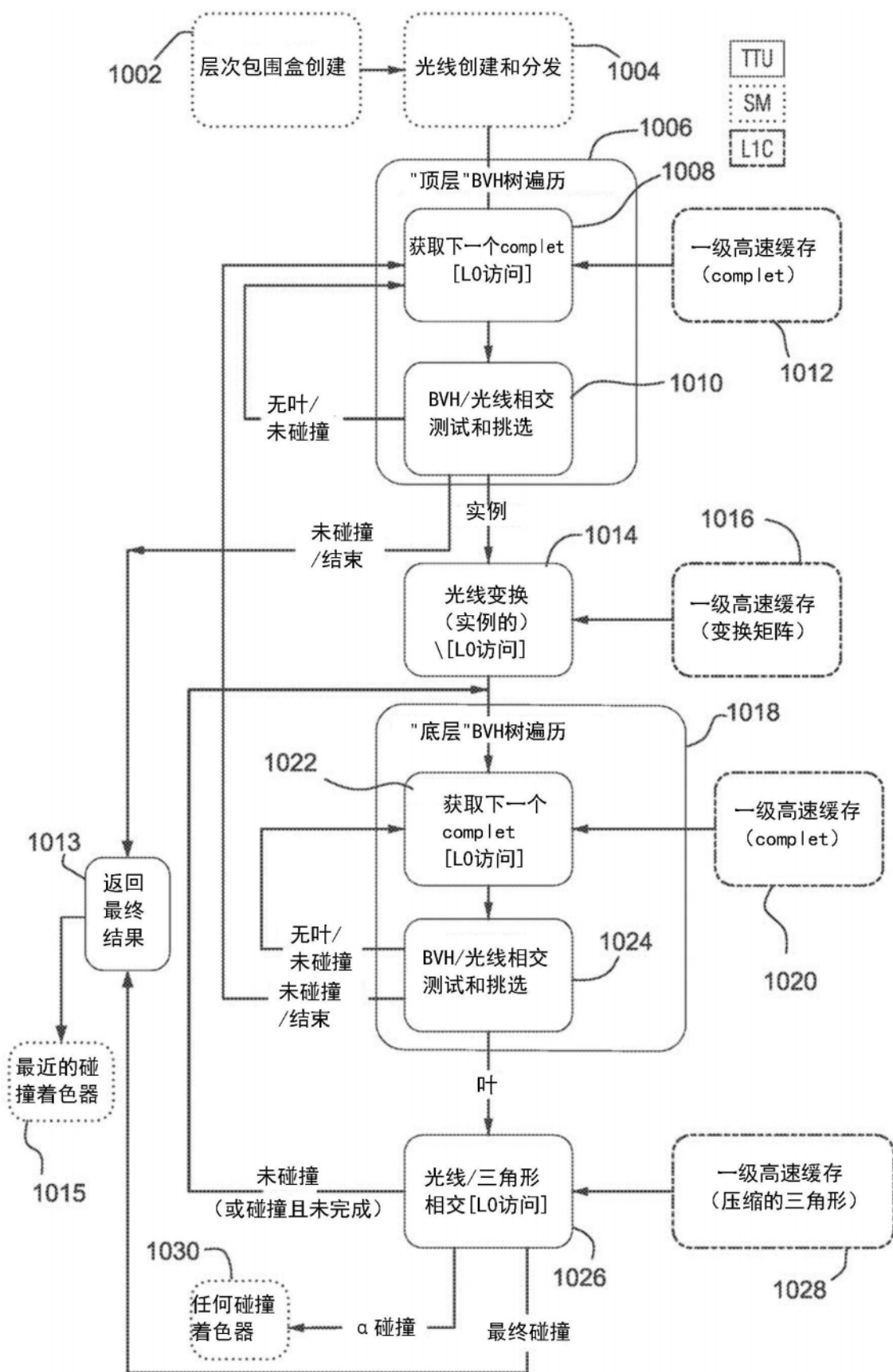


图10B

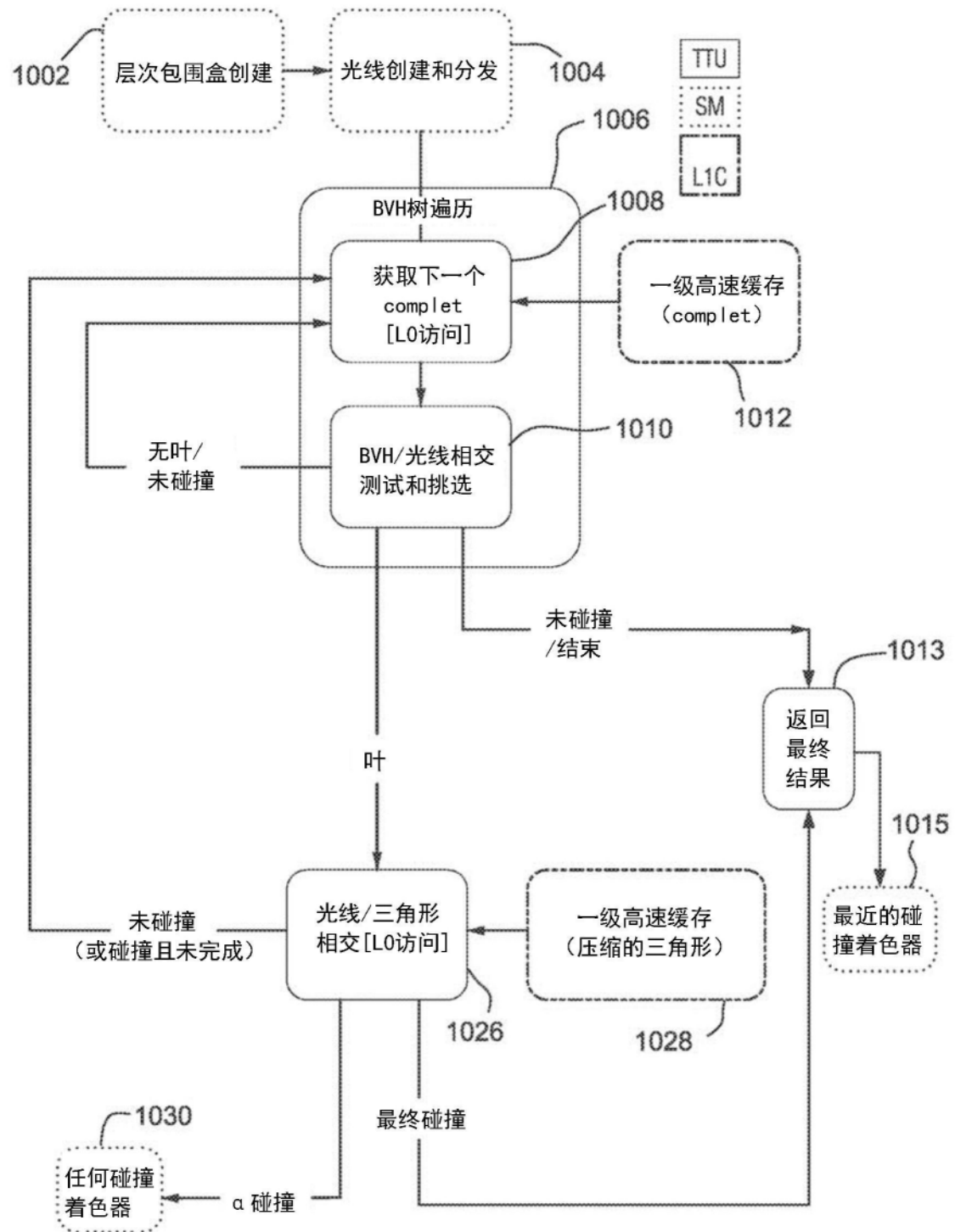


图10C

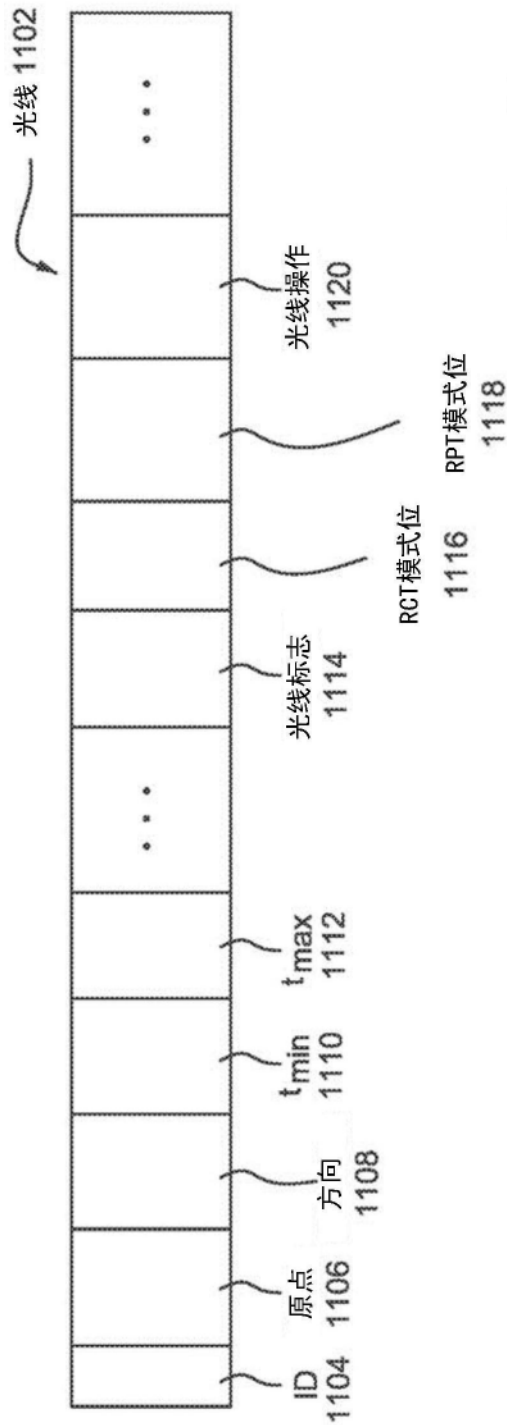


图11A

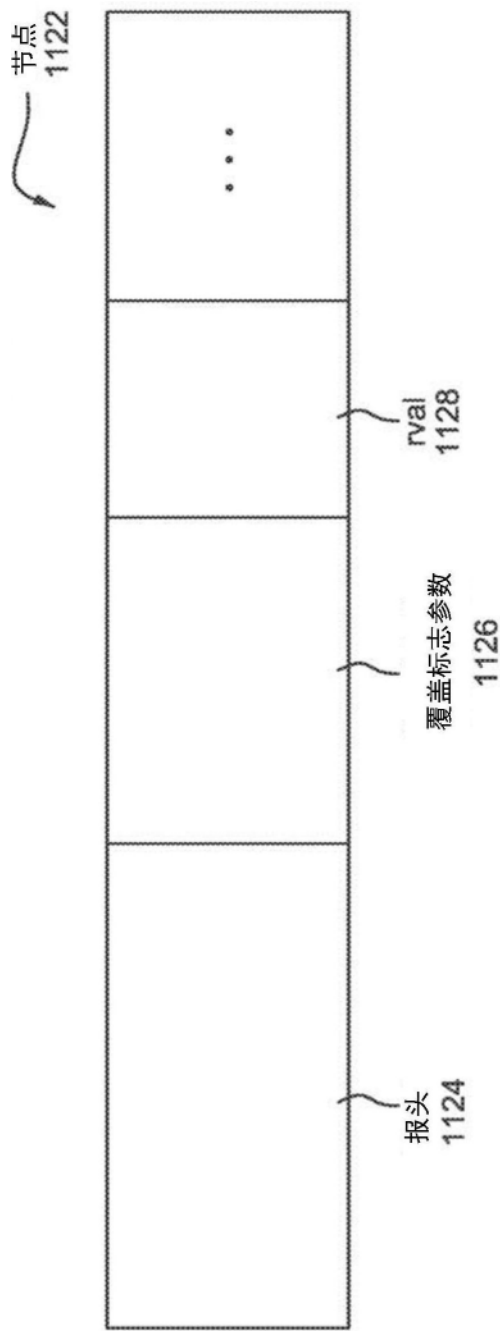


图11B

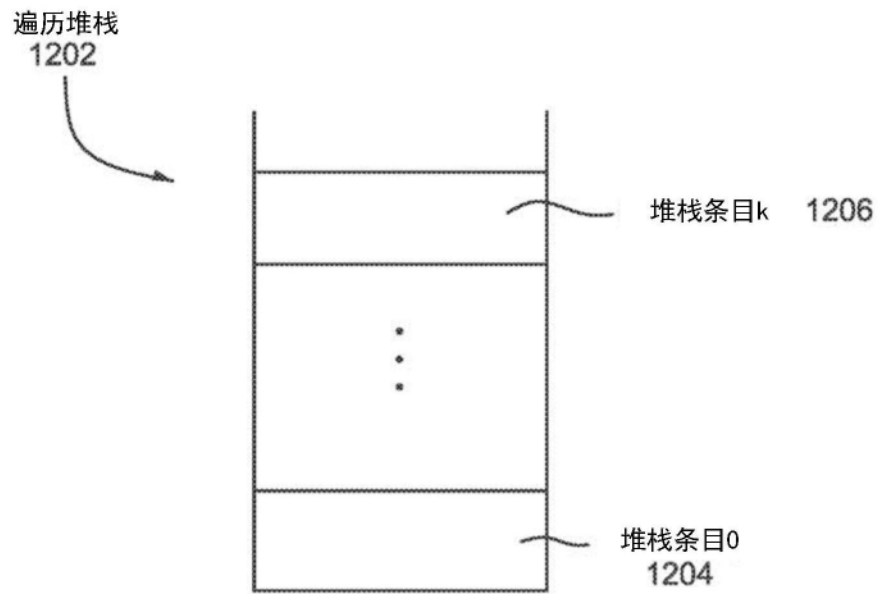


图12

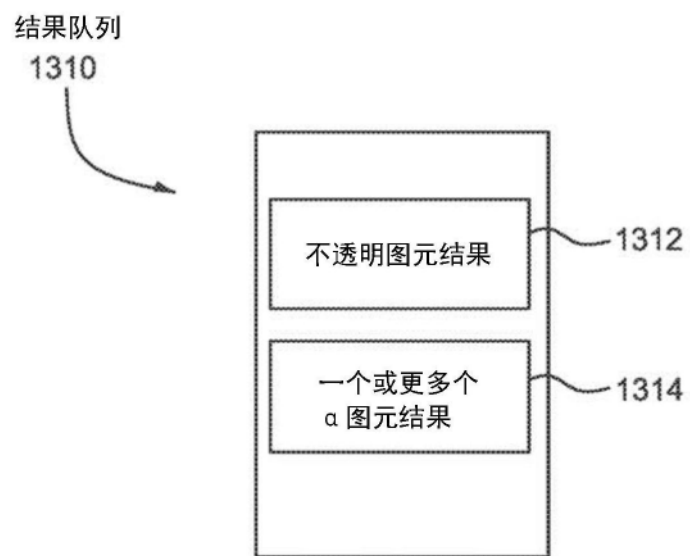


图13

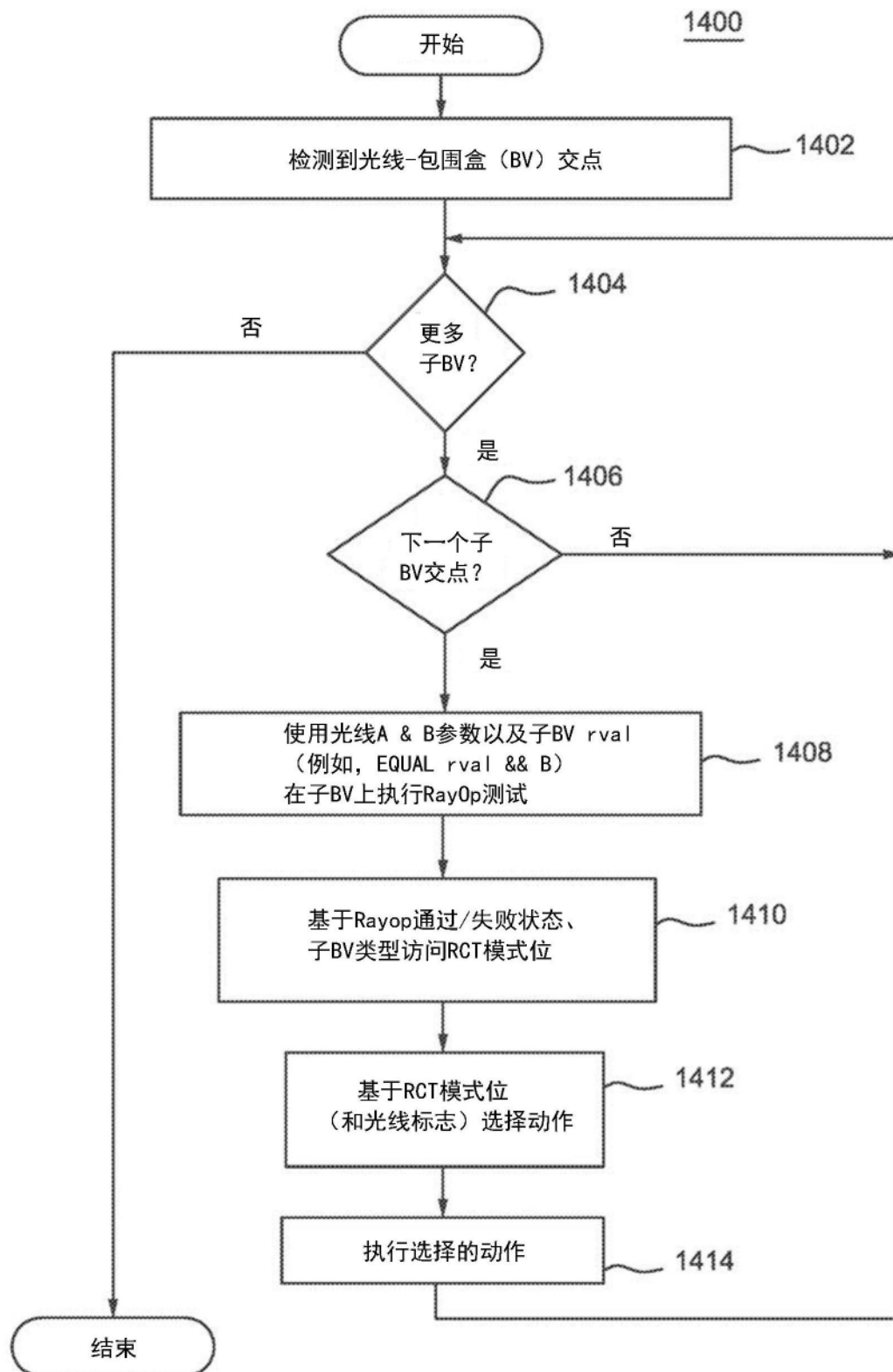


图14

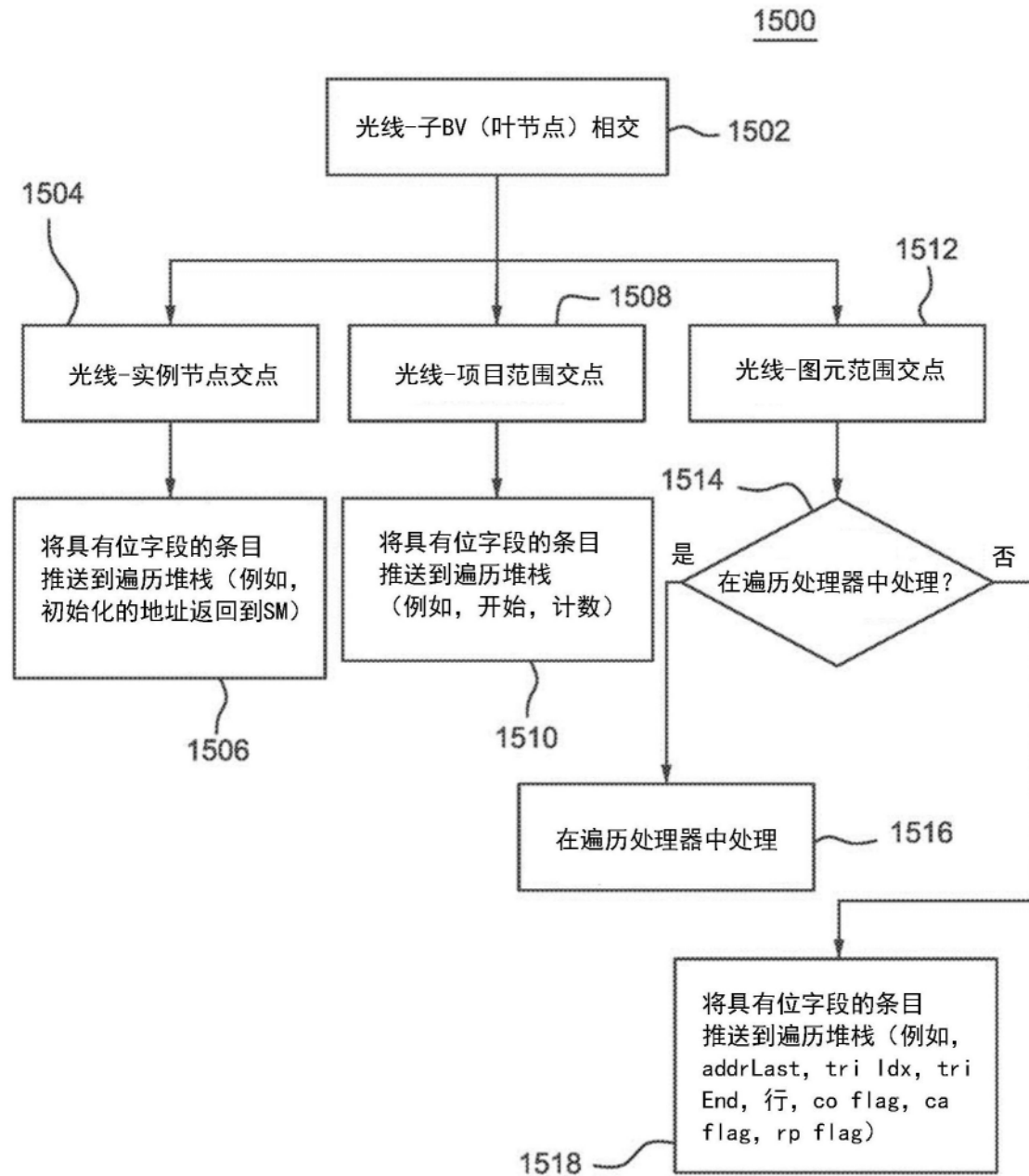


图15

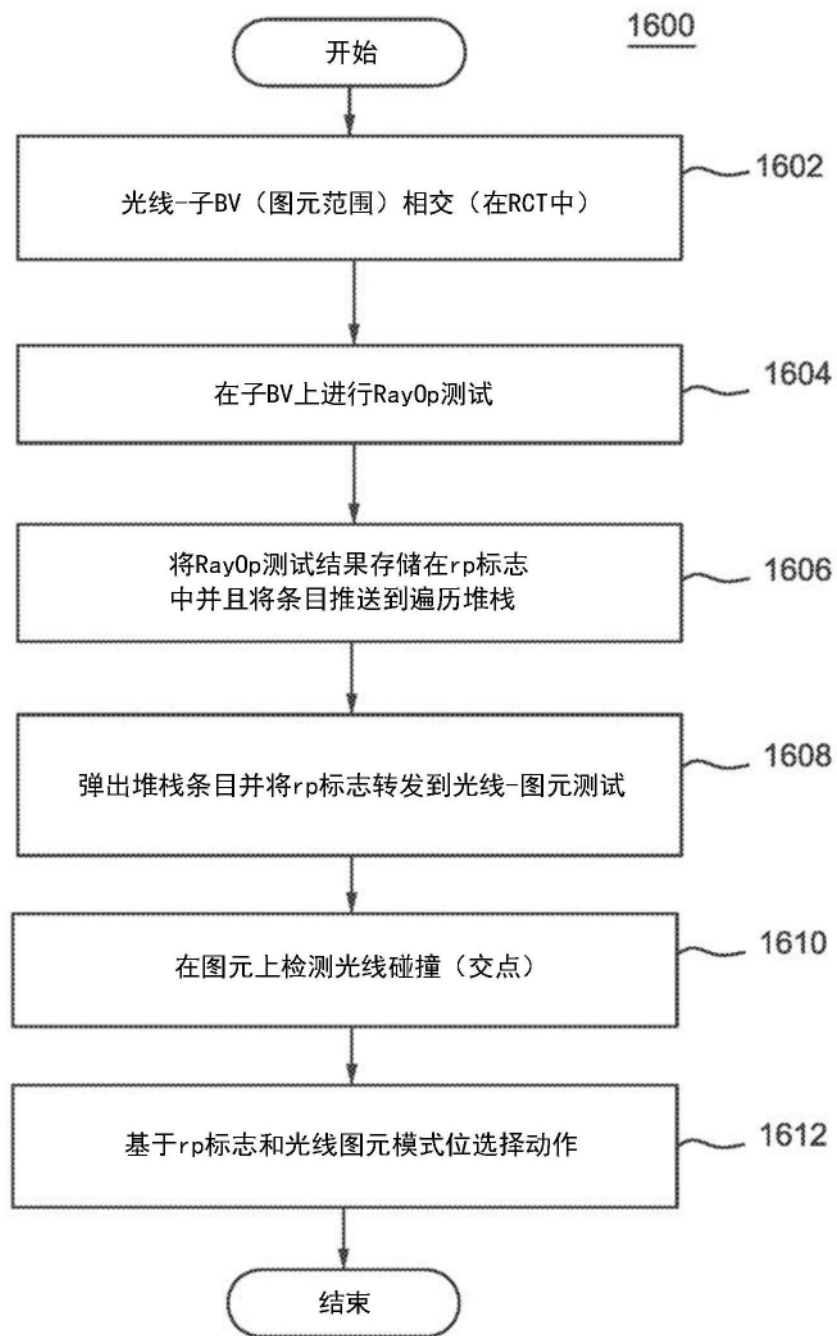


图16

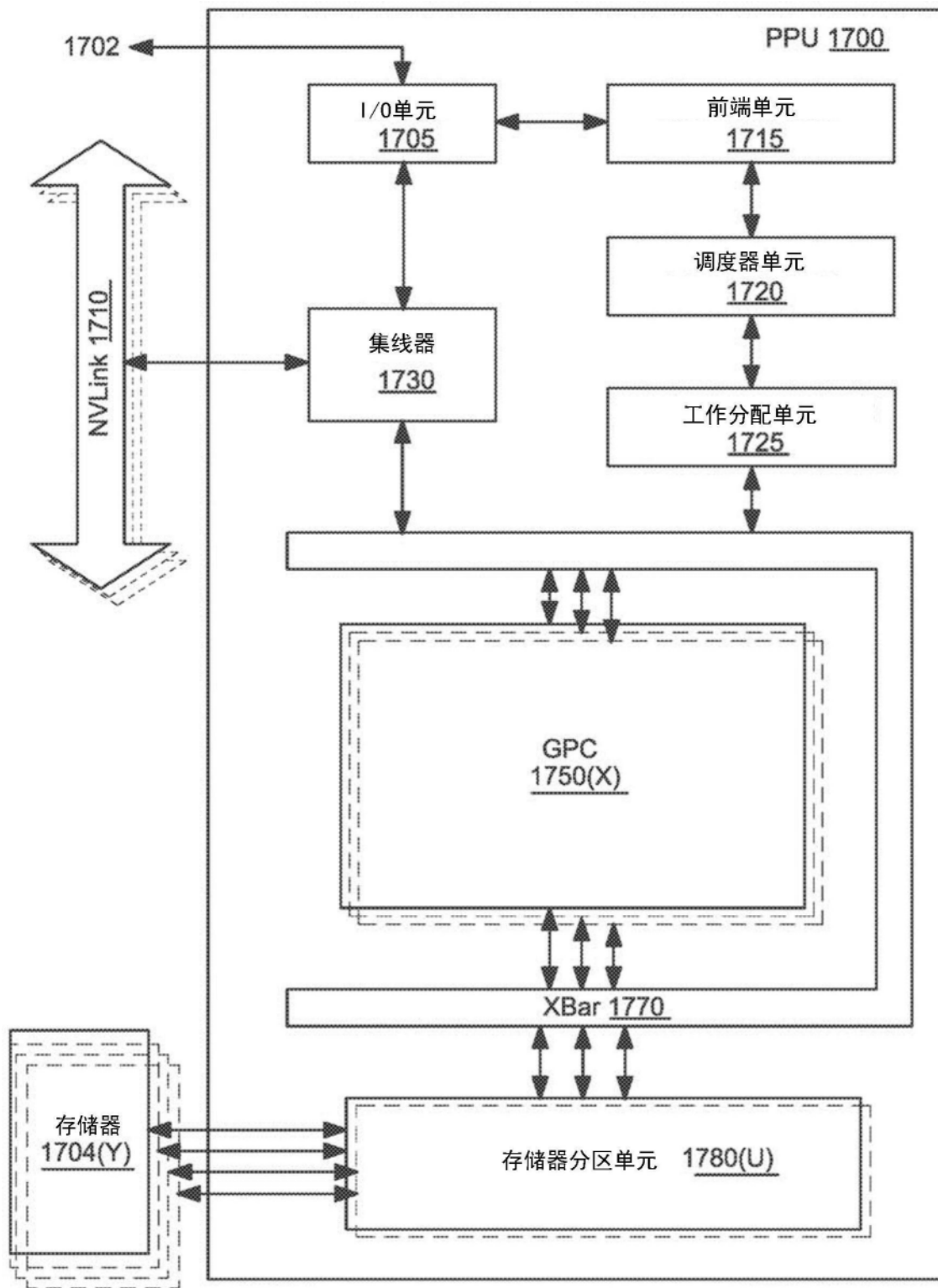


图17

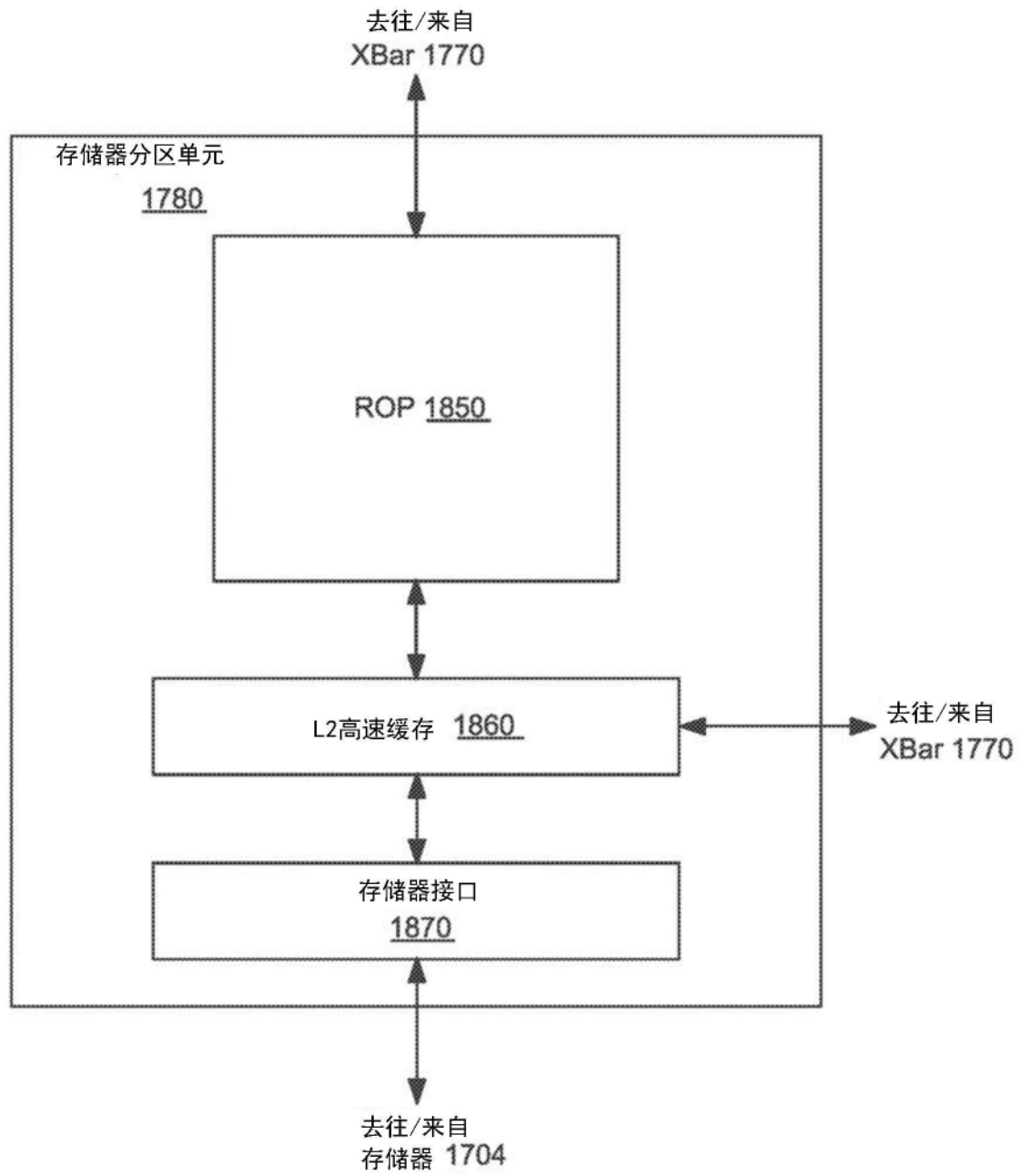


图18

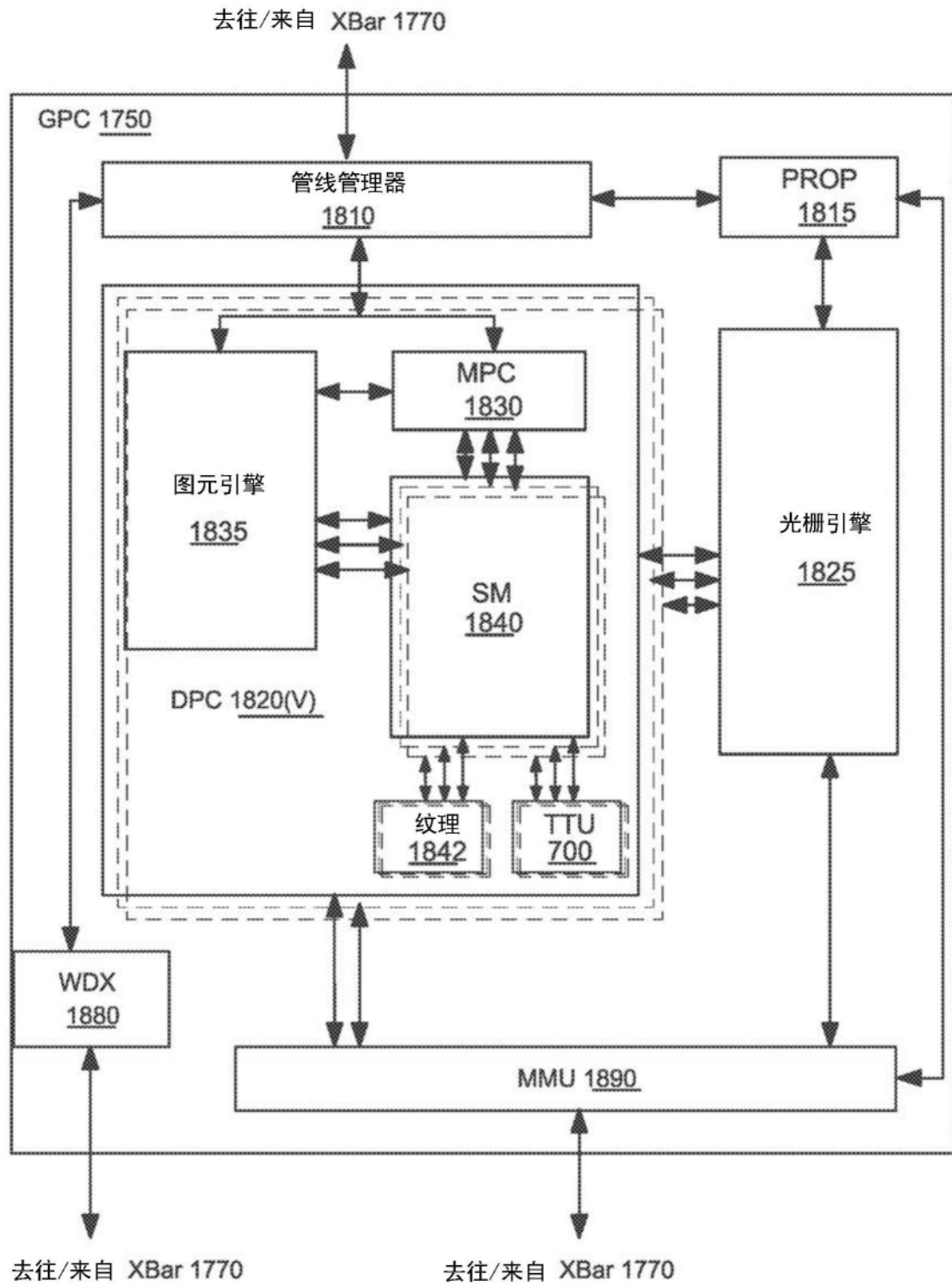


图19

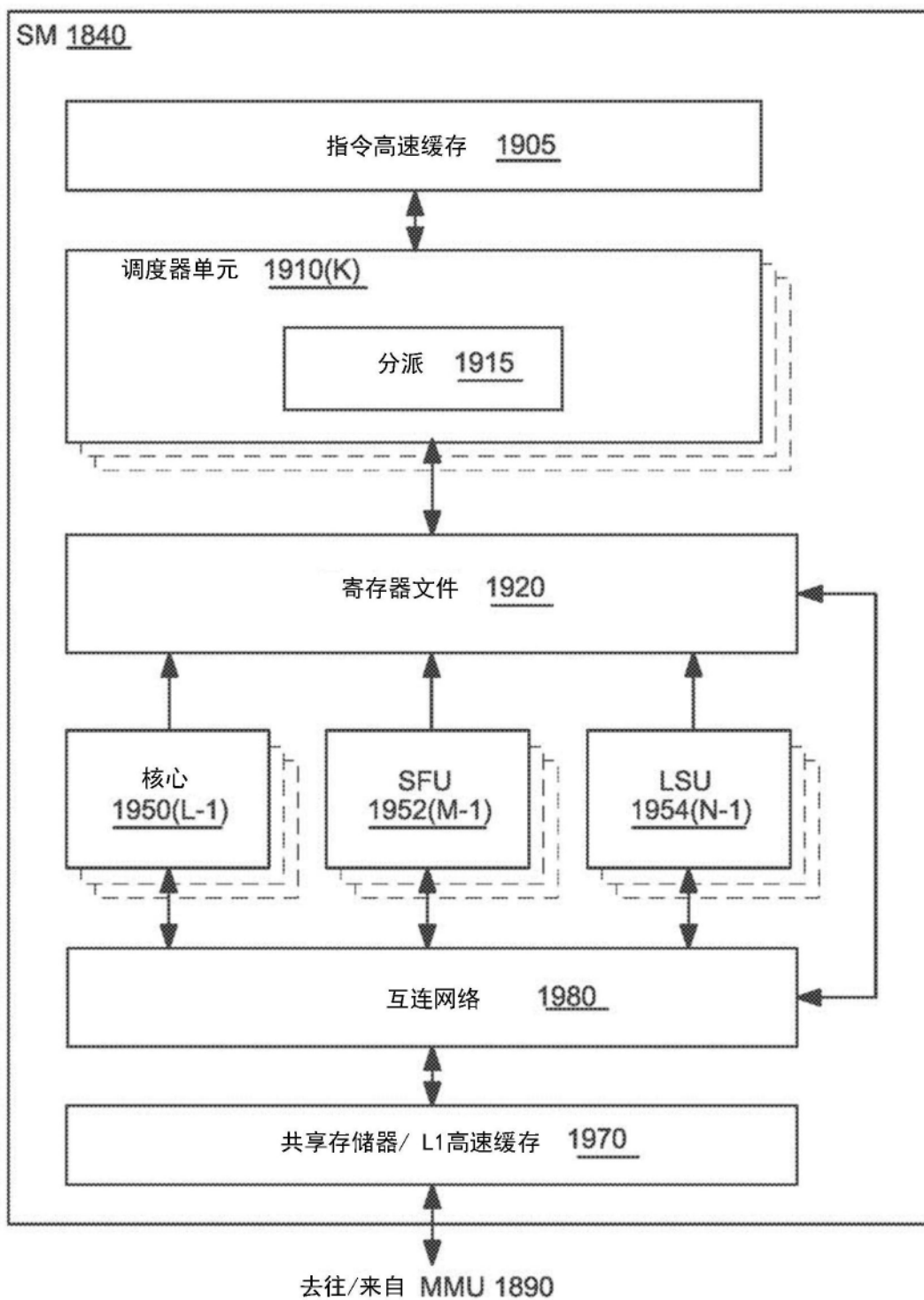


图20



图21

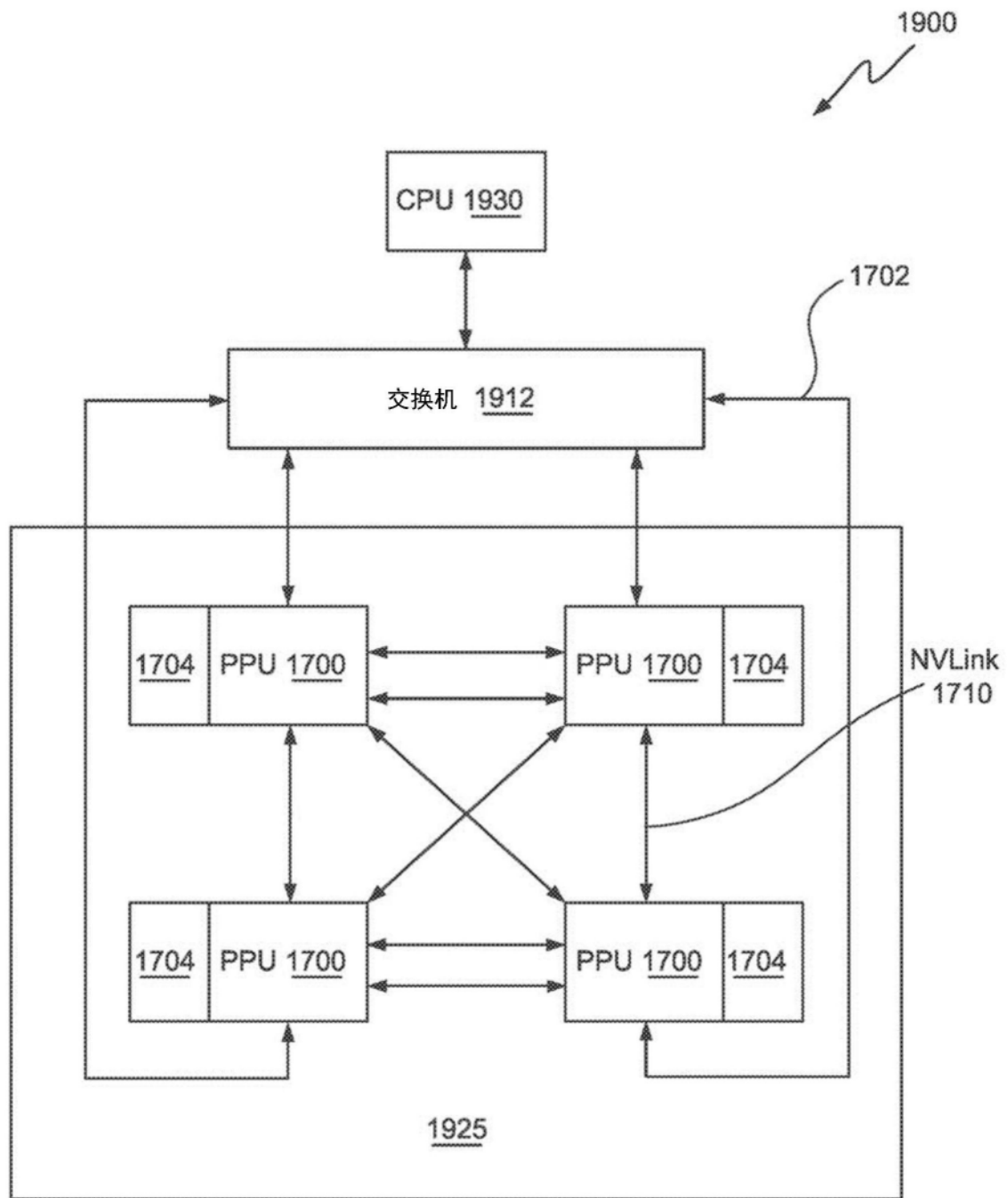


图22

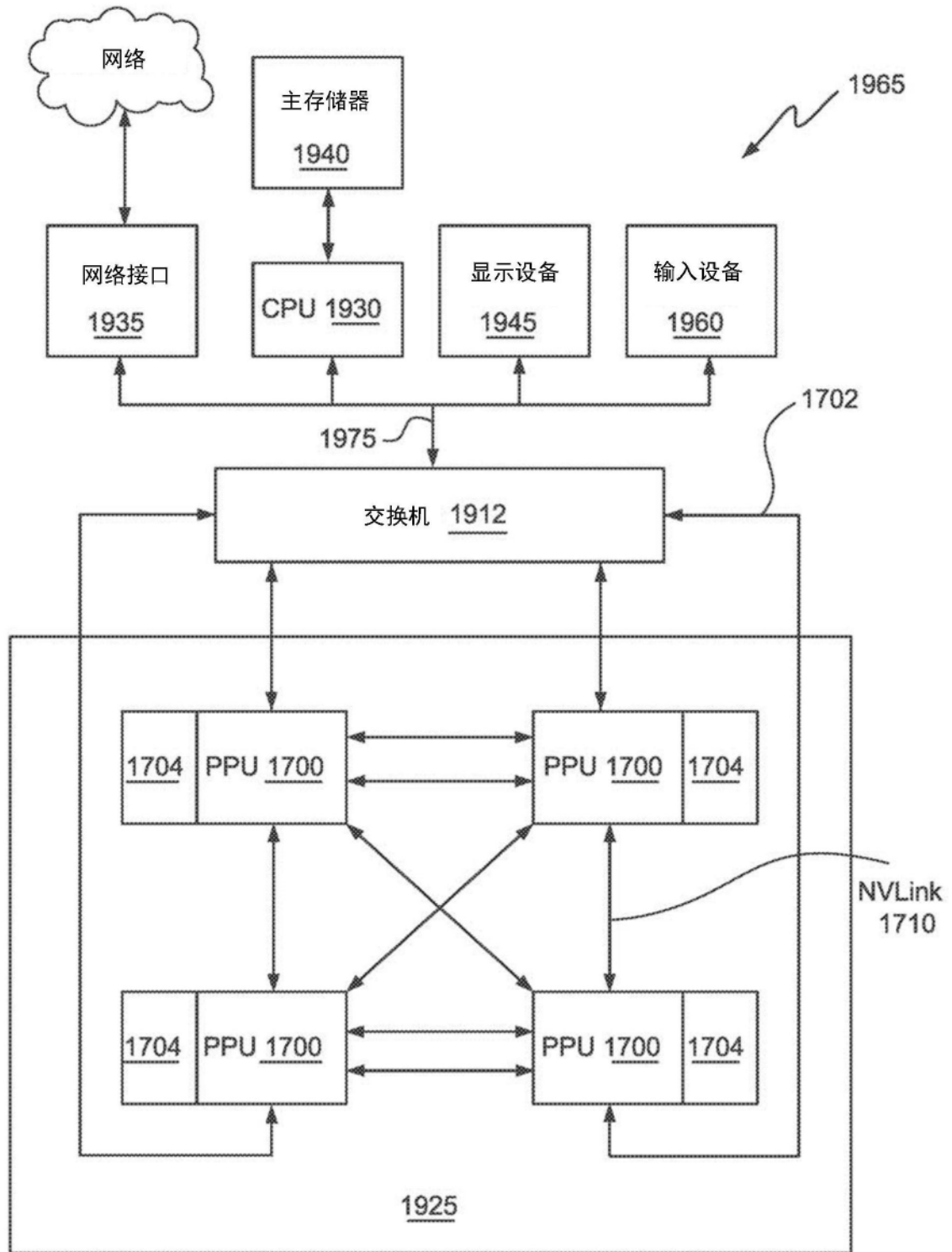


图23

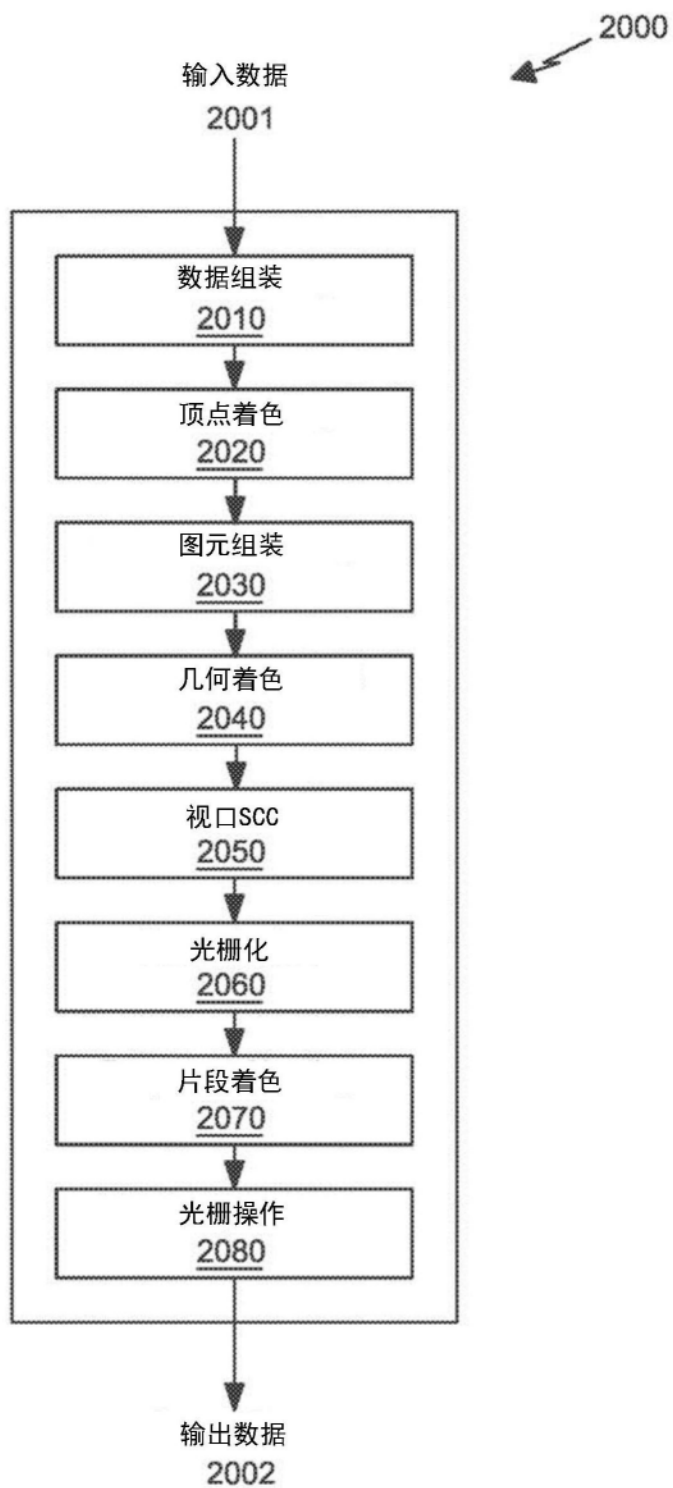


图24