[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06T 15/00 (2006.01)

[12] 发明专利申请公开说明书

[21] 申请号 200610054927.9

[43] 公开日 2006 年 9 月 20 日

[11] 公开号 CN 1835022A

[22] 申请日 2006.2.20
[21] 申请号 200610054927.9
[30] 优先权
[32] 2005. 3.18 [33] US [31] 11/083,378
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 王宗智 W·L·柏奇
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 张政权

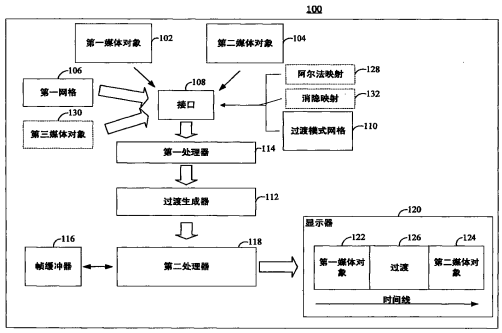
权利要求书 3 页 说明书 49 页 附图 12 页

[54] 发明名称

使用 3D 模型生成 2D 过渡

[57] 摘要

一种系统和方法用于生成第一媒体对象与第二媒体对象之间的 2D 过渡。 定义对应于期望的 2D 过渡模式的 3D 模型。 图形处理单元(GPU)或指令集根据所定义的 3D 模型生成第一媒体对象与第二媒体对象之间的过渡。 过渡生成器将第一媒体对象的多个顶点映射到第一网格创建第一经映射媒体对象，其中该第一网格与该 3D 模型相关。 该过渡生成器将第二媒体对象的多个顶点映射到过渡模式网格创建第二经映射媒体对象，其中该过渡模式网格与该 3D 模型相关。 GPU 或指令集从第一经映射媒体对象和第二经映射媒体对象呈现该过渡。



1．一种用于使用图形处理单元生成第一媒体对象与第二媒体对象之间的二维过渡的方法，所述方法包括：

定义对应于期望的二维过渡模式的三维模型；以及

使用所述图形处理单元，根据所定义的三维模型生成所述第一媒体对象与所述第二媒体对象之间的过渡。

2．如权利要求 1 所述的方法，其特征在于，还包括：

将二维坐标系统中的所述第一媒体对象的多个顶点映射给第一网格以创建第一经映射媒体对象，其中所述第一网格与所述三维模型相关；

将所述第二媒体对象的多个顶点映射给过渡模式网格以创建第二经映射媒体对象，其中所述过渡模式网格与所述三维模型相关；以及

使用所述图形处理单元从所述第一经映射媒体对象与所述第二经映射媒体对象呈现所述过渡。

3．如权利要求 2 所述的方法，其特征在于，还包括生成包含阿尔法值的阿尔法映射，且所述呈现包括根据所述阿尔法映射的运行时生成，从所述第一经映射媒体对象与所述第二经映射媒体对象和所述阿尔法映射呈现所述过渡。

4．如权利要求 3 所述的方法，其特征在于，还包括：

根据所述阿尔法映射，计算对应于沿与所述第一媒体对象和所述第二媒体对象相交的边的像素的阿尔法值；以及

将所述计算出的阿尔法值应用于所述像素以配合呈现所述第一媒体对象与第二媒体对象之间的过渡。

5．如权利要求 4 所述的方法，其特征在于，还包括：

根据消隐映射，计算对应于沿与所述第一媒体对象和所述第二媒体对象相交的边的像素的消隐阿尔法值；以及

将所述计算出的消隐阿尔法值应用于所述像素以配合呈现所述第一媒体对象与第二媒体对象之间的过渡。

6．如权利要求 1 所述的方法，其特征在于，所述过渡模式网格表示三维体积立方体，所述三维体积立方体包括 x、y 和 z 坐标参数。

7．如权利要求 1 所述的方法，其特征在于，还包括：

将第三媒体对象的多个顶点映射到所接收的过渡模式网格以创建第三经映射媒体对象；以及

从所述第一经映射媒体对象、第二经映射媒体对象与第三经映射媒体对象呈现所述过渡。

8．如权利要求 1 所述的方法，其特征在于，一个或多个计算机可读介质含有用于执行如权利要求 1 所述的方法的计算机可执行指令。

9．一种用于生成第一媒体对象与第二媒体对象之间的二维过渡的系统，所述系统包括：

用于接收所述第一媒体对象、所述第二媒体对象和过渡模式网格的接口；

用于将所接收的第一媒体对象的多个顶点映射到第一网格以创建第一经映射媒体对象的过渡生成器，其中所述过渡生成器将所接收的第二媒体对象的多个顶点映射到所述过渡模式网格以创建第二经映射媒体对象；以及

用于从所述第一经映射媒体对象与所述第二经映射媒体对象呈现所述第一媒体对象与第二媒体对象之间的过渡的图形处理器。

10．如权利要求 9 所述的系统，其特征在于，所述图形处理器从所述第一经映射媒体对象、第二经映射媒体对象和阿尔法映射呈现所述第一媒体对象与第二媒体对象之间的过渡，所述阿尔法映射包括对应于位于沿所述第一经映射媒体对象与第二经映射媒体对象相交处的边的多个像素的阿尔法值。

11．如权利要求 9 所述的系统，其特征在于，所述过渡模式网格表示三维体积立方体，其中所述三维体积立方体包括 x、y 和 z 坐标参数。

12．如权利要求 9 所述的系统，其特征在于，所述过渡模式网格的 z 坐标参数表示所述呈现进度的函数。

13．如权利要求 9 所述的系统，其特征在于，所述接口还接收第三媒体对象，且其中所述图形处理器还从所述第一经映射媒体对象、第二经映射媒体对象与第三媒体对象呈现所述第一媒体对象与第二媒体对象之间的过渡。

14．一种用于生成第一媒体对象与第二媒体对象之间的过渡的系统，所述系统包括：

用于呈现所述第一媒体对象与第二媒体对象的第一处理器或第一指令集；

用于接收所述第一媒体对象、所述第二媒体对象和过渡模式网格的接口；

用于将所接收的第一媒体对象的多个顶点映射到第一网格以创建第一经映射媒体对象的过渡生成器，所述过渡生成器将所接收的第二媒体对象的多个顶点映射

到所接收的过渡模式网格以创建第二经映射媒体对象；以及

独立于所述第一处理器或第一指令集的第二处理器或第二指令集，用于从所述第一经映射媒体对象与所述第二经映射媒体对象呈现所述第一媒体对象与第二媒体对象之间的过渡。

15．如权利要求 14 所述的系统，其特征在于，所述第二处理器或第二指令集从所述第一经映射媒体对象、第二经映射媒体对象和阿尔法映射呈现所述第一媒体对象与第二媒体对象之间的过渡，所述阿尔法映射包括对应于位于沿所述第一经映射媒体对象与第二经映射媒体对象相交处的边的多个像素的阿尔法值。

16．如权利要求 14 所述的系统，其特征在于，还包括用于为所述第二处理了器存储以下至少其中之一的帧缓冲器：顶点数据、纹理数据或模板数据。

17．如权利要求 14 所述的系统，其特征在于，所述过渡模式网格表示三维体积立方体，其中所述三维体积立方体包括 x、y 和 z 坐标参数。

18．如权利要求 14 所述的系统，其特征在于，所述接口还接收消隐映射，且其中所述第二处理器或第二指令集还从所述第一经映射媒体对象、所述第二经映射媒体对象与所述消隐映射中的消隐阿尔法值呈现所述第一媒体对象与第二媒体对象的过渡。

使用3D模型生成2D过渡

技术领域

本发明的实施例一般涉及图形动画和处理领域，尤其涉及使用三维模型和图形处理单元的处理能力生成二维过渡。

背景技术

在典型的图形处理中，软件应用程序使用硬件接口（例如，图形设备接口（GDI））来显示或打印位图化的文本（例如，TrueType字体）、图像和其它图形元素。例如，GDI负责通过调用适当的显示驱动程序并向它们传递关于将要绘制的项的信息来以一致的样式在屏幕上绘制项（诸如对话框、按钮和其它图形元素）。为利用GDI，软件程序一般使用一组例程，诸如在软件开发工具包（SDK）库中提供的那些例程以将这些软件应用程序配置成调用GDI函数。例如，SDK库包括诸如图标操作、矩形操作等函数。

在当前的计算机系统和配置下，大多数GDI特征是由中央处理单元执行的。结果是，除了执行其它进程以外，CPU需要处理图形元素和图像，诸如呈现2D图像、视频流等。因为图形处理需要对图形参数的大量计算，诸如像素或纹理计算、对图像的渲染或动画显示等，额外的图形处理妨碍了CPU和计算机系统的总体性能。尽管CPU足够能处理这些图形或图像，但仍期望对图形或图像处理的改进。

另外，大多数当前的系统或应用程序使用二维算法来处理图形特效。例如，用户可在两种媒体对象或图像中间插入过渡，使得显示该两种媒体对象的结果序列具有平滑的过渡。在现有系统中，除了显示该结果序列以外，CPU 需要资源和时间来执行该算法和程序设计指令。

从而，期望存在这样一种系统和方法，它们用于使用3D模型和图形处理单元（GPU）的固定的流水线函数将图形处理任务委托或移交给GPU以创建二维图形特效。或者，期望用于使用在3D坐标系统中处理二维特效的指令集来创建二维图形特效的系统和方法。

发明内容

本发明的实施例通过使用三维坐标系统实现二维过渡克服了现有技术的一个或多个缺点。取代将与二维过渡相关的经变换的数据发送给 CPU 以执行矩阵计算、像素或纹理计算，本发明的实施例将数据发送给 GPU，而不将它们发送给 CPU，使得 GPU 可以更有效地处理数据。GPU 可将媒体对象、动画或其它图形元素呈现在视频缓冲器上以供准备显示。或者，计算机可执行指令集或软件应用程序可在 3D 坐标系统中处理数据以创建二维特效。因此，本发明的实施例减少了总线通信量、CPU 负载，并协助缩短处理过渡、动画或其它图形变换的呈现时间。

另外，本发明的实施例在媒体对象或图像之间作出某种过渡（例如，消隐、淡入淡出或简单的擦除过渡），从而占据更少的 CPU 时间，并使得开发新的视觉变换和过渡模式更有效。该设计通过混合擦除模式、呈现技术和参数向用户提供视觉变化的丰富集合。本发明的实施例也使用 GPU 固定功能流水线以执行和处理三维媒体对象，同时在目前市场上的多数机器上产生二维图形特效。或者，本发明的实施例可使用指令集或软件应用程序的固定功能的流水线来处理三维媒体对象以产生二维图形特效。

根据本发明的一个方面，一种方法使用图形处理单元在第一媒体对象与第二媒体对象之间生成二维过渡。定义了对应于所期望的二维过渡模式的三维模型。使用图形处理单元根据所定义的三维模型生成第一媒体对象与第二媒体对象之间的过渡。

根据本发明的另一方面，一种系统生成第一媒体对象与第二媒体对象之间的二维过渡。接口接收第一媒体对象、第二媒体对象和过渡模式网格。过渡生成器将所接收的第一媒体对象的多个顶点映射至第一网格以创建第一经映射媒体对象。过渡生成器将所接收的第二媒体对象的多个顶点映射至过渡模式网格以创建第二经映射的媒体对象。图形处理器从第一经映射媒体对象和第二经映射媒体对象呈现第一媒体对象于第二媒体对象之间的过渡。

根据本发明的又一方面，一种系统生成第一媒体对象与第二媒体对象之间的过渡。第一处理器或第一指令集呈现第一媒体对象和第二媒体对象。接口接收第一媒体对象、第二媒体对象和过渡模式网格。过渡生成器将所接收的第一媒体对象的多个顶点映射到第一网格以创建第一经映射媒体对象。过渡生成器将所接收的第二媒体对象的多个顶点映射到所接收的过渡模式网格以创建第二经映射的媒体对象。独立于第一处理器或第一指令集，第二处理器与第二指令集用于从第一经映射媒体

对象和第二经映射媒体对象呈现第一媒体对象与第二媒体对象之间的过渡。

或者，本发明可包括各种其它方法和装置。

其它特征部分是明显的，部分将在后文中指出。

附图说明

图 1 是根据本发明的一个实施例，示出用于生成在第一媒体对象与第二媒体对象之间的二维过渡的系统的框图。

图 2A 是根据本发明的一个实施例，示出三维模型的示意图。

图 2B 是根据本发明的一个实施例，示出正交投影变换的示意图。

图 2C 是根据本发明的一个实施例，示出不同的维度中的过渡模式的视图的图示。

图 3A-F 是根据本发明的一个实施例，示出过渡模式的实现的示意图。

图 4A-C 是根据本发明的一个实施例，使用三维模型的圆形擦除过渡的实现的图示。

图 5A-B 是根据本发明的一个实施例的示例性过渡模式的实现的图示。

图 6 是示出根据本发明的一个实施例，用于在第一媒体对象与第二媒体对象之间使用过渡模式的锐边生成二维过渡的方法的流程图。

图 7 是示出根据本发明的一个实施例，用于对第一媒体对象与第二媒体对象之间的过渡创建额外的特效的方法的流程图。

图 8 是示出可在其中实现本发明的合适的计算系统环境的一个示例的框图。

图 9A-D 是示出根据本发明的一个实施例的阿尔法映射的示例的示意图。

在所有附图中，相应的参考标号指示相应的部分。

具体实施方式

首先参考图 1，一框图示出了根据本发明的一个实施例，用于在第一媒体对象 102 与第二媒体对象 104 之间生成二维（2D）过渡的系统 100。过渡是根据模式对从第一媒体对象到第二媒体对象的显示上的逐渐改变。例如，假设第一媒体对象 102 是白色图像，而第二媒体对象 104 是黑色图像，用户想要从该白色图像过渡到该黑色图像。过渡的一种实现可以是从左至右擦除过渡，其中黑色图像从左至右逐渐替换白色图像，直到黑色图像完全可视且完全替换了白色图像。这样的从左至右擦除过渡可包括诸如这样的模式：垂直线从左至右水平地跨过该白色图像，并当该

垂直线逐渐从左向右移动时在该垂直线的左侧引入黑色图像，直到该黑色图像完全替换该白色图像。可以实现诸如垂直移动的水平线、从中心移动的两根垂直线等模式。

系统 100 可以是如图 8 中所示的系统 130 或包含第一处理器 114、第二处理器 118、显示器 120 和存储（未示出）的其它计算机化的系统。在另一实施例中，系统 100 包括用于操纵和显示三维（3D）对象的应用程序编程接口（API）。例如，这样的 API 可以是微软公司的 Direct3D®、苹果公司的 QuickDraw® 3D、或 Silicon Graphics 有限公司的 OpenGL®。第一处理器 114 可以是 CPU、其它中央或主处理器、第一计算机可执行指令集或用于处理的软件应用程序。第二处理器 118 可以是 GPU、其它辅助处理器、第二计算机可执行指令集或用于处理涉及 3D 模型的数据的软件应用程序。

最初，在显示器 120 上显示第一媒体对象 102 和第二媒体对象 104。例如，第一媒体对象 102 或第二媒体对象 104 可以是静止图像、视频流或将显示在显示器 120 上的其它图形元素（例如，图标、按钮、网页等）的文件或文件集合。用户可能希望在时间线中创建第一媒体对象 102 与第二媒体对象 104 之间的平滑或其它有趣的过渡。除了创建有趣的过渡之外，用户可能希望观众聚焦在通过媒体对象描述的故事上而非过渡和/或特效上。同样，用户可能想要创建具有微妙特效和变化的过渡。例如，假设用户希望给予观众刺激，或与观众分享荣誉，诸如呈现奖杯等，那么用户可能想要通过创建第一媒体对象 102 与第二媒体对象 104 之间的过渡以此宣告该奖杯的出场以便奖杯具有盛大的出场。另一方面，用户可能在电影场景的结束处对过渡取景器生成微妙过渡（诸如，软边、消隐边和/或花边过渡）。

由接口 108 接收第一媒体对象 102 与第二媒体对象 104。例如，接口 108 可以是软件应用程序的图形用户界面（GUI）或是接收第一媒体对象 102 与第二媒体对象 104 的其它接口装置。第一媒体对象 102 和第二媒体对象 104 可以是通过诸如通信网络（例如，系统总线、因特网、内联网等）的直接传输从非易失性存储、易失性存储等对接口 108 的输入。

在一个示例中，用户可创建第一媒体对象 102 与第二媒体对象 104 之间的擦除过渡。尽管擦除过渡是 2D 图形特效（例如，第二媒体对象 104 从左至右跨过计算机显示器以替换第一媒体对象 102），本发明的实施例为任何给定过渡定义了 3D 模型（例如，x、y、z 坐标系统）。例如，图 2A 是根据本发明的一个实施例示出 3D 模型 200 的框图。模型 200 显示具有三个坐标轴（x 轴、y 轴和 z 轴）的 3D 视

图空间 204。在另一实施例中，3D 视图空间 204 被配置成适用于显示器 120 的不同屏幕高宽比（例如，4x3 或 16x9）。模型 200 也示出眼睛 202 显示观众正沿着与 z 轴相反的方向（由箭头 212 示出）观看。换言之，z 轴指向观众。

现在参考图 2B，示意图示出根据本发明的一个实施例用于将 2D 媒体对象映射至 3D 模型的正交投影变换。变换 214 是具有沿着水平边的 x 轴和与 x 轴正交的 y 轴的 2D 正方形。在一个实施例中，变换 214 的长和宽被置为 1 个单位，使得可视范围为 0 到 1。即，变换 214 是 1x1 单位正方形。可以理解，可使用其它形状或多边形（例如，三角形、矩形等）用于将 2D 形状映射到 3D 模型。z 轴指向位于 x-y 平面上方的眼睛 216。如图 2B 中所示示例，眼睛位于位置（0.5, 0.5, 1.0）（例如，从 x-y 平面上方观看）。换言之，2D 媒体对象可以被映射至变换 214（即，0,0->1.0, 1.0 空间），使得该 2D 媒体对象可以由与 3D 模型中的 3D 坐标系统相关的数据表示。这样的数据可由 GPU、计算机可执行指令集或软件应用程序处理来创建 2D 图形特效。

图 2C 是根据本发明的一个实施例，过渡模式在不同维度中的视图的图示。作为示例，视图 206、208、210 描述了不同视图空间中的车轮过渡。透视侧视图 206 是如由 x-y-z 坐标符号表示的 3D 模型中的车轮过渡。在使用 3D 变换的众多现有系统中，透视图 206 通过使得接近人眼的对象比远离人眼的对象更大来模拟人类视野。另一方面，正交视图 208 在如由 x-y 坐标符号表示的 2D 中将车轮过渡压平为俯视图中的椭圆。这是当车轮过渡作为 2D 过渡从第一媒体对象 102 过渡至第二媒体对象 104 来显示时的示例性视图。另一方面，自 z 轴的过渡模式网格视图 210 是应用过渡模式网格 110 之后该车轮过渡的视图。

为了进一步描述过渡模式网格视图 210，图 3A-F 是示出根据本发明的一个实施例的过渡模式网格 110 的实现的示意图，诸如从第一媒体对象 304 的中心使用第二媒体对象 306 替换第一媒体对象 304 的擦除过渡。例如，如图 3A 中所示，第一媒体对象 304 是线条填充的图像，而第二媒体对象 306 是白色的图像。从由眼睛 302 表示的方向观看的用户希望从中心将第二媒体对象 306 引入第一媒体对象 304。最初，如图 3D 中的 308 所示，用户只看见第一媒体对象 304。在图 3B 中，擦除过渡模式网格被映射到第一媒体对象 304 上来形成第一经映射媒体对象 304'。例如，第一媒体对象 304 的每一顶点或像素点被变换或对准在该 3D 中的过渡模式网格之上，使得 3D 中的过渡模式网格上的每一像素点匹配 2D 中的第一媒体对象 304 上的相应顶点或像素点。在一个实施例中，第二媒体对象 306 被映射至第一网格

106 以形成第二经映射的媒体对象 306'，以下将描述第一网格 106。

仍参考图 3B，随擦除过渡的进行，第一经映射媒体对象 304'朝着第二媒体对象 306'移动，使得用户看见如图 3E 中 308'所示。换言之，用户开始看见第一媒体对象 304 到第二媒体对象 306 的过渡。在图 3C 中，如图 3F 中的 308"所示，当该过渡过程完成时，用户仅看见结果的第二媒体对象 306"，而第一经映射媒体对象 304"对用户不可见。

在一个实施例中，在过渡过程中，第二媒体对象 306'朝着第一经映射媒体对象 304'移动。在另一实施例中，第一经映射媒体对象 304'和第二媒体对象 306'在过渡过程中朝向彼此移动。同样，，尽管对本领域中的技术人员是已知的，通过使用 3D 中的过渡模式网格，2D 过渡是由 GPU 或软件应用程序而不是 CPU 处理的，使得这些过渡被更有效地处理。

回到图 1，接口 108 也接收第一网格 106 和过渡模式网格 110，媒体对象被映射至过渡模式网格 110 使得可呈现媒体对象之间的过渡。例如，第一网格 106 可以是用于映射第一媒体对象 102 的顶点以形成第一经映射媒体对象的全屏矩形网格。换言之，第一媒体对象 102 的每一顶点被映射到 3D 坐标中的第一网格 106 上，使得 GPU 或软件应用程序可处理或计算对应于每一顶点的像素值以供显示。使用图 2B 作为另一示例，变换 214 宽 1 单位，长 1 单位，从而创建了 1x1 的正方形。如图 2B 中所示，2D 坐标系统中变换 214 的四个角中的每一个可被表示为(0,0)、(1,0)、(1,1)和(0,1)。在将第一媒体对象 102 映射至第一网格 106 时，第一媒体对象 102 的每一像素点或顶点被对准或变换至变换 214。同样，变换的 z 坐标可置为 0 以表示平面 3D 对象。

在一个实施例中，第一网格 106 包括与纹理坐标相关的第一媒体对象 102 的顶点数据（例如，与 3D 纹理空间中的单个颜色值相关的纹理值）。在另一实施例中，每一顶点的纹理坐标值等于 x、y 坐标值的值，即(tu, tv) = (x, y)，使得纹理的四个角被映射至第一网格 106 的四个角。在另一实施例中，使用 Gouraud 着色模式，且纹理值被应用于填充第一网格 106。附图 A 的第 2 节描述创建第一网格 106 的示例性实现。对于关于第一网格和/或过渡模式网格实现的信息，在 MICROSOFT® 网站的 MICROSOFT® 开发者网络 (MSDN®) 子域上提供的 MICROSOFT® Direct3D®的整个“DirectX® Software Development Kit（DirectX®软件开发工具包）”通过引用包含在此。

在另一实施例中，第一网格 106 定义了一组顶点和一组多边形面。当接收第

一网格 106 和第一媒体对象 102 之后，过渡生成器 112 将第一媒体对象 102 映射在第一网格 106 上以形成第一经映射媒体对象。同样，过渡模式网格 110 包括由用户选取的用于从第一媒体对象 102 过渡到第二媒体对象 104 或进行相反过渡的过渡模式。例如，接口 108 在显示器 120 上显示诸如具有各种形状的擦除过渡等各种过渡的示例，且用户可使用输入设备（例如，鼠标或键盘）来选取所期望的过渡模式。

图 4A-C 是根据本发明的一个实施例，使用 3D 模型在擦除过渡模式中圆形淡出和圆形淡入的实现的图示。首先，图 4A 显示作为过渡模式的圆形淡出擦除过渡的透视侧视图。第一媒体对象 402 的进入（在由圆形过渡模式映射之后）被显示为向外出或第二媒体对象 418 的过渡。例如，一开始，（或当 z 坐标值为 0 附近时），第一媒体对象 402 的进入将被显示为外出或第二媒体对象 418 上的小圆形。在该圆形淡出擦除过渡中，用户看见从屏幕或显示器 120 中心显现的圆形。随过渡的进行，该圆形越来越大。进入或第一媒体对象 402 在该圆形内显示，而外出或第二媒体对象 418 随圆形变得更大而由进入或第一媒体对象 402 取代。同样，进入或第一媒体对象 402 的 z 坐标表示过渡进度。随 z 坐标值增加，该点或该小圆形将其半径从小增加到较大，使得外出或第二媒体对象 418 被过渡到进入或第一媒体对象 402。在一个实施例中，观察空间被限于取景器的显示（例如，显示器 120），因此，该圆形过渡的一部分由于未由显示器 120 显示而被裁剪。

类似地，示出进入或第二媒体对象 404 和外出或第一媒体对象 422 的圆形淡入擦除过渡的另一透视侧视图。圆形淡入擦除过渡随进入或第二媒体对象 404 过渡到外出或第一媒体对象 422 而以较大的圆形开始变成点，而不是最初以点开始（或当 z 坐标值接近 0 时）。

为了进一步说明，图 4B 显示圆形擦除过渡的三个视图。屏幕截图 406 显示使用圆形淡出擦除的第一媒体对象 A 过渡到第二媒体对象 B 的快照。正交视图 408 显示 2D 视图中的圆形，而透视侧视图 410 显示锥形的圆形。同样，本发明的实施例将圆形过渡应用于诸如模型 200 等 3D 模型，使得 2D 圆形被转换成锥形。通过在 3D 模型中使用过渡模式网格 110，本发明的实施例将用于 2D 过渡的与 3D 模型相关的数据发送给第二处理器或第二指令集 118（共同表示为“第二处理器 118”），使得可更有效地处理 2D 过渡。

图 4C 示出了根据本发明的一个实施例，圆形淡出和圆形淡入过渡的进行的另一示例。例如，框 412 描绘了作为外出媒体对象的第一媒体对象 414 通过圆形淡出过渡过渡到作为进入媒体对象的第二媒体对象 416。在过渡进度 0%处，框 412 仅

示出第一媒体对象 414。在进度 25%处，当圆形淡出过渡引入第二媒体对象 416 时，第二媒体对象 416 开始显示。当进度增加至 100%，框 412 示出了如何显示 2D 圆形淡出擦除过渡。类似地，框 420 描绘了作为外出媒体对象的第一媒体对象 414 通过圆形淡入擦除过渡而过渡到第二媒体对象 416。

可以理解，尽管使用不同模式的擦除过渡的用户体验在本领域中可能是已知的，但使用 GPU 或软件应用程序的使用 3D 过渡模式处理 2D 过渡更有效且使用更少的 CPU 处理资源。

也可以理解，如图 4A 至 4C 中所示的过渡模式（例如，圆形淡入或圆形淡出擦除过渡）作为示例而非限制示出。例如，可实现其它过渡模式，诸如泵状（swipe）、对角线向下、star、车轮状、蝴蝶结、圆形、矩形、分割、展现及插入、光圈、V 填充、锯齿形、棋格、对角框、对角十字、菱形、雷达、多星形、多圆形、锁孔状、心形、扫（向内、向上、向外、向下）、扇形（向内、向上、向外、向下）、眼状或其它过渡，而不背离本发明的精神。作为额外的示例，见图 5A 和 5B，它们是根据本发明的一个实施例的示例性过渡模式的实现的图示。图 5A 示出在 30%进度的矩形过渡。图 5A 也描述了正交俯视图和透视侧视图。类似地，图 5B 描述了 30%进度处的锯齿形过渡、该过渡模式的正交俯视图和透视侧视图。此外，附录 A 的第 1 节描述了使用 3D 模型创建过渡模式的实现。

类似于形成第一经映射媒体对象，过渡生成器 112 在从接口 108 接收第二媒体对象 104 和过渡模式网格 110 之后，将过渡模式网格 110 映射到第二媒体对象 104 的顶点以创建第二经映射的媒体对象。该第一经映射的媒体对象和第二经映射的媒体对象被发送给第二处理器 118 用于在显示棋 120 上从第一经映射媒体对象 122 和第二经映射媒体对象 124 呈现第一媒体对象 102 与第二媒体对象 104 之间的过渡 126。例如，图 4C 中的框 412 示出了使用第二处理器或第二指令集 118 从第一经映射媒体对象 414 和第二经映射媒体对象 416 呈现圆形淡出过渡的结果。

在一个实施例中，系统 100 也包括帧缓冲器 116，用于为第二处理器 118 临时存储顶点、纹理、模板或其它数据以供呈现过渡 126。例如，第二处理器 118 可按照以下呈现顺序处理擦除过渡：（1）如果需要，将过渡模式网格呈现给掩模；（2）将第一网格 106 呈现给帧缓冲器 116；以及（3）将过渡模式网格 110 呈现给帧缓冲器 116。例如，在每一步，第二处理器 118 维护该序列中的呈现状态，使得在显示下一媒体对象或过渡之前，系统 100 可对所显示的媒体对象或过渡应用恰当的呈现状态。在另一示例中，帧缓冲器 116 可以是易失性存储、非易失性存储或其它存

储介质中的视频存储器。

在另一实施例中，第二处理器 118 创建临时对象以协助呈现过渡 126。例如，在呈现过渡（例如，擦除过渡）过程中，第二处理器 118 可创建临时对象以访问另外的硬件和软件功能，诸如加速光栅化和硬件/软件顶点处理。在又一实施例中，第二处理器 118 创建额外的临时对象以处理第一和过渡模式网格、阿尔法和消隐混合（将在以下描述）。例如，第二处理器 118 创建临时对象以创建或加载第一网格 106、过渡模式网格 110、阿尔法和消隐混合，使得这些对象可从硬件或软件调用或执行特定功能。在第二处理器 118 呈现过渡 126 或其它处理之后，可从存储中移除或摧毁这些临时对象。

在一个实施例中，本发明的实施例随过渡创建附加的特效。例如，图 3B-C 和图 4A-B 中所示的过渡将过渡模式的锐边示为从第一媒体对象改变成第二媒体对象的过渡模式。图 6 是示出根据本发明的一个实施例用于使用过渡模式的锐边生成第一媒体对象与第二媒体对象之间的 2D 过渡的方法的流程图。最初，在 602 处，系统 100 加载第一网格（例如，第一网格 106）和过渡模式网格（例如，过渡模式网格 110）。例如，在用户从由软件应用程序提供的或从其它来源供应的过渡模式集合中选取之后，接口 108 从用户处接收过渡模式网格（例如，圆形淡入、锯齿形等）。在 604 处，系统 100 定义用于生成所选取的 2D 过渡的 3D 模型。例如，如变换 214 所定义，通过使用模型 200 和正交变换 214，系统 100 为从 0.0 到 1.0 的行进定义了动画循环。

在 606 处，接口 108 接收第一媒体对象，作为外出媒体对象（例如，第一媒体对象 304）。在 608 处，过渡生成器 112 将第一媒体对象映射至第一网格以形成第一经映射媒体对象。在 610 处，接口 108 接收第二媒体对象（例如，第二媒体对象 306），在 612 处过渡生成器 112 将第二媒体对象映射至从接口 108 所选取/接收的过渡模式网格（见图 3A-G 作为映射实现的示例的描述）。第二处理器 118 接着在 614 处从第一经映射媒体对象和第二媒体对象呈现第一媒体对象与第二媒体对象之间的过渡（例如，过渡 126）。例如，在呈现中，第二处理器 118 在过渡发生之前首先计算与第一经映射媒体对象和第二媒体对象的顶点、像素和/或纹理相关的数据。第二处理器 118 接着根据进度计算数据，即，随着过渡 126 在将第一媒体对象 102 改变成第二媒体对象 104 的过程中进行时。在完成计算数据之后，第二处理器 118 将计算出的数据转化为将在显示器 120 上显示的格式。例如，显示 120 可以是具有高宽比 4x3 的计算机监视器，第二处理器 118 可将计算出的数据转化为

适于在具有高宽比 4x3 的显示器 120 上显示过渡 126。结果是，在呈现之后，显示器 120 显示从第一媒体对象 102 过渡到第二媒体对象 104 的过渡 126。参见附录 A 的第 3 节，它描述了创建锐边的一种实现。

如所述，本发明的实施例除使用 3D 模型生成过渡之外，还创建额外的特效以在媒体对象之间过渡。例如，本发明的实施例可在当第一媒体对象与第二媒体对象相交的过渡过程中软化边缘。

为软化现有技术产生的硬边或锐边，可使用阿尔法混合技术。例如，接口 108 接收阿尔法映射 128，用于软化过渡模式的锐边。在一个实施例中，第二处理器 118 接收和处理参数，诸如每一图元（材质阿尔法）参数、每一顶点（顶点阿尔法）参数、每一像素阿尔法（纹理、帧缓冲器和呈现目标阿尔法）参数等阿尔法混合参数。通过阿尔法混合技术，阿尔法值围绕第一媒体对象与第二媒体对象相交处的边缘逐渐减小。在一个实施例中，使用阿尔法映射以存储阿尔法值。

图 9A-D 是示出根据本发明的一个实施例对阿尔法映射 900 的使用的示例的示意图。图 9A 中的对象 904 显示在 30%过渡进度处的对角线十字过渡，而图 9B 中的对象 906 显示了在 70%过渡进度处的同一对角线十字过渡。在 30%进度处，沿线 902 的阿尔法值被选取来定义过渡进度。例如，点 922 处的阿尔法值为图 9A 中的对象 904 中的两个相应的点 912 和 914 定义了过渡进度值。类似地，点 950 处的阿尔法值为图 9A 中的对象 904 中的相应点 908 定义了过渡进度值。可以理解，阿尔法映射 900 中的阿尔法值可用于表示在呈现第一媒体对象 102 和第二媒体对象 104 的过程中的一个或多个相应的点。也可理解，尽管图 9A-C 是使用黑白色着色来描绘的，但可使用其它颜色的着色和/或颜色纹理的程度，而不背离本发明的范围。

在 70%进度处，沿线 910 的阿尔法值被选取来定义过渡进度。例如，点 924 处的阿尔法值为图 9B 中的对象 906 中的两个相应点 916 和 918 定义了过渡进度值。类似地，在点 952 处的阿尔法值为图 9B 中的对象 906 中的相应点 920 定义了过渡进度值。可以理解，随过渡的进行，指示过渡进度的另一条线将指示从阿尔法映射 900 的何处选取阿尔法值。

图 9D 示出了根据本发明的一个实施例，阿尔法映射 926 中的渐变半透明区域。阿尔法映射 926 包括三个区域：不透明区域 938、透明区域 940 和其中第一媒体对象 102 与第二媒体对象 104 混合在一起的渐变半透明区域（由虚线框定义）。在阿尔法映射 926 中，点 928 处的阿尔法值包括对应于 100%白色和 0%黑色的颜色着

色的值，在点 932 处的阿尔法值包括对应于 0%白色和 100%黑色的颜色着色的值。划分阿尔法映射 926 的对角线 944 显示了 50%的白色着色和 50%的黑色着色。5%宽度的区域（由虚线围成的）示出了其中可选取对应于 50%白色和 50%黑色的颜色着色的阿尔法值的区域。例如，区域 930 和 942 中的阿尔法值可具有对应于 50%白色的值。在一个实施例中，渐变半透明区的宽度可在 0%到 50%的范围之内。例如，在 0%宽度的渐变半透明区处，不存在渐变半透明区，且作为结果，阿尔法映射 926 中的阿尔法值包括对应于 100%白色或 0%白色的值中的两者任一。或者，在 50%宽度处，阿尔法映射 926 中的阿尔法值将全部包括对应于 50%白色的值。通过定义阿尔法映射 926 的渐变半透明区的不同宽度，本发明的实施例可选择阿尔法映射 926 中的阿尔法值以产生二维过渡中的软边特效。附录 A 的第 5 节描述了阿尔法映射的示例性实现。

图 7 是根据本发明的一个实施例，示出用于创建作为第一媒体对象与第二媒体对象之间的过渡的一部分的附加特效的方法的流程图。图 7 描述了用于使用阿尔法混合技术的本发明的替换实施例。可以理解，图 7 也可共用步骤 602 至 612。在替换实施例中，在 714 处接收阿尔法映射。结果是，第二处理器 118 从第一经映射媒体对象、第二经映射媒体对象和阿尔法映射呈现第一媒体对象 102 和第二媒体对象 104 之间的过渡 126。在该情形中，第二处理器 118 除上述呈现第一经映射媒体对象和第二经映射媒体对象之外还呈现阿尔法映射。使用图 4B 的屏幕截图 406 作为示例，第一经映射媒体对象的纹理或着色与第二经映射媒体对象的纹理或着色不同。在使用阿尔法映射呈现第一经映射媒体对象与第二经映射媒体对象时，第二处理器 118 通过根据阿尔法映射，计算对应于沿着与第一媒体对象与第二媒体对象相交的边的像素的阿尔法值，来软化第一和第二经映射对象相交处的边。由于第二处理器 118 根据进度从第一经映射媒体对象和第二经映射媒体对象 104 计算数据，因此计算出的阿尔法值被应用来替换沿相交边的像素。当第二处理器 118 完成应用像素时，第二处理器 118 转化除从第一经映射媒体对象和第二经映射媒体对象以外的像素给显示器 120 以供显示。附录 A 的第 4 节描述了软化第一媒体对象与第二媒体对象的相交部分之间的边缘的实现。

在另一替换实施例中，将消隐混合技术应用于第一媒体对象与第二媒体对象之间的边相交。例如，该技术要求由第二处理器 118 进行的两个呈现进程。第二处理器首先呈现过渡模式网格 110 以生成含有消隐的阿尔法值的消隐映射。在第二呈现进程中，第二处理器 118 使用消隐映射呈现第一经映射的媒体对象和第二经映射

媒体对象。换言之，类似于使用阿尔法映射呈现，第二处理器 118 生成与第一经映射媒体对象和第二媒体对象一起呈现的含有消隐的阿尔法值的分离的消隐映射。

在另一实施例中，阿尔法混合技术和消隐混合技术均可以使用。例如，可如同在阿尔法混合技术中一样从阿尔法映射中检索阿尔法值。接着，创建具有消隐的阿尔法值且使用来自阿尔法映射的阿尔法值计算的消隐映射以制成该消隐混合特效。例如，通用应用 ADDSIGNED（带符号加法）运算（例如，ADDSIGNED 计算 Alpha 1（来自阿尔法映射） + Alpha 2（来自消隐映射） - 0.5）和将结果限制在 0 与 1 之间来进行该计算。在另一实施例中，仅有具有大于 0.25 的计算出的阿尔法值的像素被显示为过渡的一部分。在该示例中，尽管这些像素未被显示，但仍存储计算出的阿尔法值以供其它使用（例如，将它们存储在模板缓冲器中）。附录 A 的第 8 节描述了消隐边缘技术的实现。

或者，在一个实施例中，在 718 处，接口 108 接收第三媒体对象。例如，除阿尔法混合、消隐混合或软边技术以外，用户可创建包含不同于第一媒体对象和/或第二媒体对象的纹理或着色的纹理或着色的边。同样，在 720 处，过渡生成器 112 将第三媒体对象映射到过渡模式网格以形成第三经映射媒体对象。在 716 处，第二处理器 118 从第一经映射媒体对象、第二经映射媒体对象、第三经映射媒体对象呈现第一媒体对象与第二媒体对象之间的过渡。附录 A 的第 6 节描述了使用第三媒体对象创建过渡的实现。

在又一实施例中，第二处理器 118 呈现包含各种技术的过渡，诸如阿尔法混合、消隐混合、软边、硬边、第三媒体对象混合或上述技术的组合。例如，用户可通过第三媒体对象混合与锐边技术的组合来创建过渡以控制最终将显示哪些像素。在该示例中，过渡生成器 112 除映射第二媒体对象以形成第二经映射媒体对象以外，还映射第三媒体对象以形成第三经映射媒体对象。接着，过渡生成器 112 将第二经映射媒体对象置于第三经映射媒体对象后方（例如，在负 z 坐标方向按照当前动画进度的量）。即，第二经映射媒体对象的顶点或像素值被映射到第三经映射媒体对象的顶点或像素值后方，使得用户仅可看见第三经映射媒体对象的像素。接着，过渡生成器 112 使用第三经映射媒体对象以替换第一媒体对象与第二媒体对象相交部分处的像素。换言之，与之前所述的阿尔法或消隐混合技术不同，第三媒体对象混合技术使用第三媒体对象来替换沿第一媒体对象 102 和第二媒体对象 104 相交部分处的边的像素的纹理值。因此，当第二处理器 118 使用第三媒体对象呈现第一媒体对象与第二媒体对象之间的过渡时，用户在显示器 120 上沿第一媒体对象与第

二媒体对象相交部分处的边看见第三媒体对象。在一个实施例中，第三经映射媒体对象的宽度一开始较窄，随过渡 126 的动画进行变得愈来愈宽。附录 A 的第 7 节描述了上述过渡特效的实现。

在使用上述技术的组合的另一示例中，可使用软边技术和第三媒体对象混合技术的组合。在一个实施例中，第二处理器 118 首先呈现第二经映射的媒体对象，然后呈现第三经映射媒体对象，反之亦可。阿尔法映射的阿尔法值被用于混合第三经映射媒体对象与第一媒体对象和/或第二媒体对象相交部分处的边的像素。

图 8 示出计算机 130 形式的通用计算设备的一个示例。在本发明的一个实施例中，诸如计算机 130 的计算机适于在此处示出和描述的其它附图中使用。计算机 130 包括一个或多个处理器或处理单元 132 和系统存储器 134。在示出的实施例中，系统总线 136 将包括系统存储器 134 在内的各种系统组件耦合至处理器 132。总线 136 表示任何若干类型的总线结构中的一种或多种，包括存储器总线或存储器控制器、外围总线、加速图形端口和使用各种总线体系结构中的任一种的局部总线。作为示例，而非限制，这样的体系结构包括工业标准体系结构（ISA）总线、微通道体系结构（MCA）总线、扩展的 ISA（EISA）总线、视频电子技术标准协会（VESA）局部总线和外围部件互连（PCI）总线（也被称为 Mezzanine 总线）。

计算机 130 通常包括至少某种形式的计算机可读介质。计算机可读介质包括易失性和非易失性介质、可移动和不可移动介质，它们可以是能够被计算机 130 访问的任何可用介质。作为示例，而非限制，计算机可读介质可以包括计算机存储介质和通信介质。计算机存储介质包括以任何方法或技术实现的用于存储诸如计算机可读指令、数据结构、程序模块或其它数据等信息的易失性和非易失性、可移动和不可移动介质。例如，计算机存储介质包括，RAM、ROM、EEPROM、闪存或其它存储器技术，CD-ROM、数字多功能盘（DVD）或其它光盘存储，磁带盒、磁带、磁盘存储或其它磁性存储设备，或能用于存储所需信息且可以由计算机 130 访问的任何其它介质。通信介质通常具体化为诸如载波或其它传输机制等已调制数据信号中的计算机可读指令、数据结构、程序模块或其它数据，且包含任何信息传递介质。本领域的技术人员熟悉已调制数据信号，它使其一个或多个特征以在信号中编码信息的方式被设定或更改。有线介质，诸如有线网络或直接线连接，以及无线介质，诸如声学、RF、红外线和其它无线介质均是通信介质的示例。上述中任一个的组合也应包括在计算机可读介质的范围之内。

系统存储器 134 包括可移动和/或不可移动、易失性和/或非易失性存储器形式

的计算机存储介质。在示出的实施例中，系统存储器 134 包括诸如只读存储器（ROM）138 和随机存取存储器（RAM）140。基本输入/输出系统 142（BIOS）包含有助于诸如启动时在计算机 130 中元件之间传递信息的基本例程，它通常存储在 ROM 138 中。RAM 140 通常包含处理单元 132 可以立即访问和/或目前正在操作的数据和/或程序模块。作为示例，而非限制，图 8 示出了操作系统 144、应用程序 146、其它程序模块 148 和程序数据 150。

计算机 130 也可以包括其它可移动/不可移动、易失性/非易失性计算机存储介质。例如，图 8 示出了从不可移动、非易失性磁介质中读取或向其写入的硬盘驱动器 154。图 8 也示出从可移动、非易失性磁盘 158 中读取或向其写入的磁盘驱动器 156，以及从诸如 CD ROM 或其它光学介质等可移动、非易失性光盘 162 中读取或向其写入的光盘驱动器 160。可以在示例性操作环境下使用的其它可移动/不可移动、易失性/非易失性计算机存储介质包括，但不限于，盒式磁带、闪存卡、数字多功能盘、数字录像带、固态 RAM、固态 ROM 等。硬盘驱动器 154、磁盘驱动器 156 和光盘驱动器 160 通常由诸如接口 166 的非易失性存储器接口连接至系统总线 136。

以上描述和在图 8 中示出的驱动器或其它大容量存储设备及其相关联的计算机存储介质为计算机 130 提供了对计算机可读指令、数据结构、程序模块和其它数据的存储。例如，在图 8 中，硬盘驱动器 154 被示为存储操作系统 170、应用程序 172、其它程序模块 174 和程序数据 176。注意，这些组件可以与操作系统 144、应用程序 146、其它程序模块 148 和程序数据 150 相同或不同。操作系统 170、应用程序 172、其它程序模块 174 和程序数据 176 在这里被标注了不同的标号是为了说明至少它们是不同的副本。

用户可以通过输入设备或用户界面选择设备，诸如键盘 180 和定点设备 182（例如，鼠标、跟踪球、笔或触摸垫）向计算机 130 输入命令和信息。其它输入设备（未示出）可以包括麦克风、操纵杆、游戏垫、圆盘式卫星天线、扫描仪等。这些和其它输入设备由耦合至系统总线 136 的用户输入接口 184 连接至处理单元 132，但也可以由其它接口或总线结构，诸如并行端口、游戏端口或通用串行总线（USB）连接。监视器 188 或其它类型的显示设备也经由接口，诸如视频接口 190 连接至系统总线 136。除监视器 188 以外，计算机通常包括其它外围输出设备（未示出），诸如打印机和扬声器，它们可以通过输出外围接口（未示出）连接。

计算机 130 可使用至一个或多个远程计算机，诸如远程计算机 194 的逻辑连接在网络化环境中操作。远程计算机 194 可以是个人计算机、服务器、路由器、网络 PC、对等设备或其它常见网络节点，且通常包括上文相对于计算机 130 描述的许多或所有元件。图 8 中所示逻辑连接包括局域网（LAN）196 和广域网（WAN）198，但也可以包括其它网络。LAN 196 和/或 WAN 198 可以是有线网络、无线网络、这两者的组合等。这样的网络环境在办公室、企业范围计算机网络、内联网和全球计算机网络（因特网）中是常见的。

当在局域网环境中使用时，计算机 130 通过网络接口或适配器 186 连接至 LAN 196。当广域网环境中使用时，计算机 130 通常包括调制解调器 178 或用于通过诸如因特网等 WAN 198 建立通信的其它装置。调制解调器 178 可以是内置或外置的，它通过用户输入接口 184 或其它合适的机制连接至系统总线 136。在网络化环境中，相对于计算机 130 所描述的程序模块或其部分可以存储在远程存储器存储设备（未示出）中。作为示例，而非限制，图 8 示出了远程应用程序 192 驻留在存储器设备上。所示的网络连接是示例性的，且可以使用在计算机之间建立通信链路的其它手段。

一般而言，通过在不同时刻存储在计算机的各种计算机可读存储介质中的指令对计算机 130 的数据处理器进行编程。程序和操作系统通常分布在例如软盘或 CD-ROM 上。从那里，它们可被安装或加载至计算机的辅助存储器。在执行时，它们被至少部分加载至计算机的主要电子存储器中。当这些和其它各种类型的计算机可读介质包含结合微处理器或其它数据处理器用于实现以下描述的步骤的指令或程序时，此处所述的本发明包括这样的介质。当根据此处描述的方法和技术对计算机进行编程时，本发明也包括计算机本身。

为说明起见，诸如操作系统等程序或其它可执行程序组件在此处示为离散的框。然而，可以认识到，这样的程序和组件在不同时刻驻留在计算机的不同存储组件中，且由计算机的数据处理器执行。

尽管结合包括计算机 130 的示例性计算系统环境描述，本发明可用于众多其它通用或专用计算系统环境或配置。该计算系统环境不旨在对本发明的使用或功能的范围提出限制。而且，该计算系统环境不应被解释为对在该示例性操作环境中示出的任何一个组件或组件的组合具有任何依赖性或要求。适合在本发明中使用的公知的计算系统、环境和/或配置的示例包括，但不限于，个人计算机、服务器计算机、手持或膝上型设备、多处理器系统、基于微处理器的系统、机顶盒、可编程消

费者电子产品、移动电话、网络 PC、小型机、大型计算机、包含以上系统或设备中的任一个的分布式计算机环境等。

本发明可在诸如由一台或多台计算机或其它设备执行的程序模块等的计算机可执行指令的通用语境中描述。一般而言，程序模块包括，但不限于，例程、程序、对象、组件、数据结构等，它们执行特定任务或实现特定抽象数据类型。本发明也可以在分布式计算环境中实现，其中任务由通过通信网络连接的远程处理设备执行。在分布式计算环境中，程序模块可以位于包括存储器存储设备在内的本地和远程计算机存储介质中。

软件体系结构环境中的接口包括软件模块、组件、代码部分或计算机可执行指令的其它序列。接口包括，例如，第一模块，它访问第二模块以便为第一模块执行计算任务。在一个示例中，第一和第二模块包括诸如由操作系统提供的应用程序编程接口（API）、组件对象模型（COM）接口（例如，用于对等应用程序通信）以及可扩展标记语言元数据互换格式（XMI）接口（例如，用于 web 服务之间的通信）。

该接口可以是诸如在 Java 2 平台企业版(J2EE)、COM 或分布式 COM(DCOM)示例中的紧密耦合的同步的实现。或者另外，该接口可以是诸如在 web 服务（例如，使用简单对象访问协议）中的松散耦合的异步实现。一般而言，该接口包括以下特征的任何组合：紧密耦合、松散耦合、同步以及异步。而且，该界面可遵循标准协议、专有协议或标准和专有协议的任何组合。

此处所述的接口全部都可以是单个接口的一部分或可被实现为分离的接口或其任何组合。该接口可本地或远程地执行以提供功能。而且，该接口可包括除此处说明或描述的额外功能或更少的功能。

除非特别指出，否则此处示出和描述的方法的执行或实行的顺序是不重要的。即，除非特别指出，否则方法的各元素可按照任何顺序执行，且该方法可包括比此处揭示的那些方法更多或更少的元素。例如，可以构想，在另一元素之前、同时或之后执行或实行特定的元素在本发明的范围之内。

当介绍本发明或其实施例的元素时，冠词“一”、“一个”、“该”和“所述”旨在意味着存在一个或多个该元素。术语“包含”、“包括”、“具有”旨在是包含性的，且意味着除所列出的元素以外，还可以存在另外的元素。

鉴于以上内容，可以看到，达到了本发明的若干目的，且获取了其它有利的结果。

由于可对以上系统和方法进行各种改变，而不背离本发明的范围，因此包含在以上描述和附图中所示的所有事物旨在被解释为说明性的而非限制性的。

附录 A

第 1 节：几何生成示例代码。

基于想要生成的“Inset（插入）”擦除模式，使 25%、50%和 75%处的过渡模式作为相交部分。将它们层叠，并连接顶点，可以获得如此的模型。x 是输出的宽；Y 是输出的高，z 是过渡的进度。

```
const DWORD VERTEX_XYZ_TEX2::FVF = D3DFVF_XYZ | D3DFVF_TEX2;
m_dwNumPatternIndices = 6;
m_dwNumPatternVertices = 4;
D3DXCreateMeshFVF(              // 创建 D3DXMeshobject 的 DirectX
                                API:
m_dwNumPatternIndices / 3, // 面/三角形的个数
m_dwNumPatternVertices,    // 顶点的个数
   D3DXMESH_SYSTEMMEM,         // 在系统存储器中创建 D3DXMesh
        VERTEX_XYZ_TEX2::FVF,      // 顶点格式：XYZ 位置和 2 组
纹理坐标
        pd3dDevice,                // GPU 进程的实例
        &pSysMemMesh               // 返回 D3DXMESH 指针
);
VERTEX_XYZ_TEX2* pVertices = NULL;
pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pVertices );
//从 D3DXMesh 对象获取 VertexBuffer。
// 基于以上绘图，使用硬编码(x, y, z)坐标填充顶点。
pVertices[0].v.x   = 1.0f;
pVertices[0].v.y   = 1.0f;
pVertices[0].v.z   = 1.0f;
// 使 (t0.x, t0.y) = (x, y) 且 (t1.x, t1.y) = (z, Progress)
pVertices[0].t0.x  = pVertices[0].v.x;
pVertices[0].t0.y  = pVertices[0].v.y;
pVertices[0].t1.x  = pVertices[0].v.z;
pVertices[0].t1.y  = 0.50f; // 进度
pVertices[1].v.x   = 0.0f;
pVertices[1].v.y   = 1.0f;
```

```
    pVertices[1].v.z   = 0.0f;
    pVertices[1].t0.x  = pVertices[1].v.x;
    pVertices[1].t0.y  = pVertices[1].v.y;
    pVertices[1].t1.x  = pVertices[1].v.z;
    pVertices[1].t1.y  = 0.50f; // 进度
    pVertices[2].v.x   = 0.0f;
    pVertices[2].v.y   = 0.0f;
    pVertices[2].v.z   = 1.0f;
    pVertices[2].t0.x  = pVertices[2].v.x;
    pVertices[2].t0.y  = pVertices[2].v.y;
    pVertices[2].t1.x  = pVertices[2].v.z;
    pVertices[2].t1.y  = 0.50f; // 进度
    pVertices[3].v.x   = 1.0f;
    pVertices[3].v.y   = 0.0f;
    pVertices[3].v.z   = 1.0f;
    pVertices[3].t0.x  = pVertices[3].v.x;
    pVertices[3].t0.y  = pVertices[3].v.y;
    pVertices[3].t1.x  = pVertices[3].v.z;
    pVertices[3].t1.y  = 0.50f; // 进度

    pSysMemMesh->UnlockVertexBuffer();
    // 向 Direct3D 告知可换出该网格对象，它目前未被使用。

    WORD* pIndices = NULL;
    pSysMemMesh->LockIndexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pIndices);
    // 从 D3DXMesh 对象获取索引缓冲器。
    // 填充索引(按顺时针方向)。
    // 三角形 1
    pIndices[0] = 0;
    pIndices[1] = 1;
    pIndices[2] = 3;
    // 三角形 2
    pIndices[3] = 1;
```

```
pIndices[4] = 2;
pIndices[5] = 3;

pSysMemMesh->UnlockIndexBuffer();
// 创建 CDXUTMesh 对象以在系统存储器和设备存储器中管理 D3DXMesh 对象。
*ppMesh = new CDXUTMesh( L"CInsetPatternMesh" );

(*ppMesh)->m_pSysMemMesh = pSysMemMesh.Detach();
//使之前创建的 D3DXMesh 对象受控于 CDXUTMesh.
```

第 2 节：网格生成示例代码。

网格被创建为 DXUTMesh 对象。

每一 DXUTMesh 对象管理 2 个 D3DXMesh 对象；一个在系统存储器中（CPU 可访问），另一个在设备存储器中（GPU/显示适配器可访问）。每次一对系统存储器中的 D3DXMesh 对象进行改变，应用程序就需要将其下载至设备存储器。否则，GPU 将对每个绘制命令重用该 D3DXMesh 对象。两个 D3DXMesh 对象的内容都是相同的。它包含一个顶点缓冲器和一个索引缓冲器。

顶点缓冲器是顶点的数组。例如，擦除过渡定义哪些数据被包含在顶点中。

每一顶点含有 3 个条目，诸如以下给出的示例：

3D 坐标，（x，y，z）用于顶点位置。每一顶点位置是基于几何形状，诸如圆形、星形、心形等计算的。

2D 纹理坐标，(tu1，tv1)用于指导 GPU 或指令集在视频资源的哪里寻找顶点的红/绿/蓝（RGB）值 。在擦除过渡实例中，tu1 被分配给 x 值；tv1 被分配给 y 值。

第二个 2D 纹理坐标，(tu2，tv2)用于指导 GPU 或指令集在阿尔法映射上的哪里去寻找该顶点的阿尔法值；tu2 被分配给 z 值；tv2 将被分配给运行时间/动画时间的进度%值。

索引缓冲器是整数索引的数组。每一索引对顶点缓冲器进行索引。每 3 个索引将形成一个三角形。所有的三角形行程一个 3D 模型。

可使用如 Alias 的 Maya®的其它 3D 建模工具来创建.X 文件。当加载所创建的.X 文件时，对每一 (x，y，z)坐标的归一化可能是缩放该 3D 模型以拟合 0.0 和 1.0 范围的可视区域所必须的。

第 3 节：锐边示例代码。

```
// 设置 Direct3D 状态
// pd3dDevice 是 Direct3D / GPU 的实例的指针。
pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE, FALSE); //
不需要执行阿尔法测试。
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,FALSE);
// 不需要执行阿尔法混合。
pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, FALSE); //
不需要执行模板测试。

pd3dDevice->SetRenderState( D3DRS_ZENABLE, D3DZB_TRUE); // 允
许深度测试。
pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE,TRUE); // 将
Z 值保存在深度缓冲器中。
// 设置 Direct3D 纹理混合状态。
// Direct3D 允许至多 8 个阶段，但只需要 1 个。
pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
0 ); // 为阶段 0 使用 t0 纹理坐标。
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
// 输出 RGB 值
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_TEXTURE );
// 从纹理输出 RGB 值。
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
// 没有从阶段 0 及以上的阿尔法输出
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_DISABLE );
// 没有从阶段 1 及以上的 RGB 输出
```

```
// 绘制进入源
// 使用纹理 2 和第一网格
pd3dDevice->SetTexture(0, pTextureIn);
// pTextureIn 是存储进入视频帧的纹理对象。
// 将该纹理置为纹理混合阶段 0。
LPD3DXMESH pPlaneMesh = m_pPlaneMesh->GetLocalMesh(); //从
CDXUTMesh 获取存储在设备/GPU 存储器中的 D3DXMesh 对象。
pPlaneMesh->DrawSubset( 0 ); // 使 D3D/GPU 执行使用该 PlaneMesh、
TextureIn 以及当前设定的绘制。
// 绘制外出源
// 使用纹理 1 和过渡模式网格
// 应用平移变换矩阵将过渡模式网格移动至 z 方向较低的位置。
D3DXMATRIX matTranslation;
D3DXMatrixTranslation( &matTranslation, 0, 0, -fltProgress);
// 创建平移变换矩阵将 PatternMesh 移动至 z 方向更低位置。
pd3dDevice->SetTransform( D3DTS_WORLD, &matTranslation );
// 设置 D3D 世界变换矩阵。
// D3D 世界变换将模型空间坐标转化为世界坐标。
// 过渡模式网格中的(0, 0, 0)变成(0, 0, -fltProgress)
pd3dDevice->SetTexture(0, pTextureOut);
LPD3DXMESH pPatternMesh = m_pPatternMesh->GetLocalMesh();
pPatternMesh->DrawSubset( 0 ); // 使 D3D/GPU 执行使用该
PatternMesh、TextureOut 以及当前设置的绘制。
```

对每一帧重复该例程。基于过渡持续时间、帧速率以及当前帧计算 fltProgress。

当 D3D / GPU 绘制第一网格/进入视频源，它将整个 TextureIn 置于呈现目标表面，还为呈现目标表面中的每一像素将 z 值置于深度缓冲器中。

当 D3D / GPU 绘制过渡模式网格/外出视频源时，它将执行深度测试。它将过渡模式网格像素的 z 值与深度缓冲器中的像素的 z 值进行比较。离观众更远的像素将被丢弃。

第 4 节：软边示例代码。

```
// 首先绘制进入视频源。
// 设置 Direct3D 状态
// pd3dDevice 是 Direct3D / GPU 的实例指针。
pd3dDevice->SetRenderState( D3DRS_SHADEMODE,
D3DSHADE_GOURAUD);
//设置为平滑着色模式。因此 D3D / GPU 将计算每一顶点处的纹理坐标，然后执行每一三角形的 3 个顶点之间的内插，以计算该三角形内的每一像素的纹理坐标。稍后将使用该纹理坐标来从所选取的纹理中查找 RGB 和阿尔法值。

pd3dDevice->SetRenderState(D3DRS_ALPHATESTENABLE, FALSE); //
无需执行阿尔法测试。
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,FALSE);
// 无需执行阿尔法混合。
pd3dDevice->SetRenderState( D3DRS_STENCILENABLE,   FALSE);
// 无需执行模板测试。
pd3dDevice->SetRenderState( D3DRS_ZENABLE, D3DZB_FALSE); //
无需执行深度测试
pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE,    FALSE);
// 无需在深度缓冲器中保存 z 值

// 设置 Direct3D 纹理混合状态。
// Direct3D 允许至多 8 个阶段，但仅需要 1 个。

//从 pTextureIn 的(tu1, tv1)为每一顶点查找 RGB 值。
pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
0 ); // 为阶段 0 使用 tu1、tv1 纹理坐标。
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARG1); // 输出 RGB 值
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_TEXTURE ); // 从纹理中输出 RGB 值
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
```

```
D3DTOP_DISABLE ); // 没有从阶段 0 及以上的阿尔法输出
   pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_DISABLE );
   // 没有从阶段 1 及以上的 RGB 输出。

   // 绘制进入源
   // 使用纹理 2 和第一网格

   pd3dDevice->SetTexture(0, pTextureIn);
   // pTextureIn 是存储进入视频帧的纹理对象。
   // 将该纹理置为纹理混合阶段 0。

   LPD3DXMESH pPlaneMesh = m_pPlaneMesh->GetLocalMesh(); // 从
CDXUTMesh 获取存储在设备/GPU 存储器中的 D3DXMesh 对象
   pPlaneMesh->DrawSubset( 0 ); // 使 D3D/GPU 执行使用该 PlaneMesh、
TextureIn 和当前设定的绘制

   // 绘制外出源

   // 设置呈现状态
   // 用于阿尔法混合操作的设置。
   pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
   pd3dDevice->SetRenderState( D3DRS_SRCBLEND,
D3DBLEND_SRCALPHA );
   pd3dDevice->SetRenderState( D3DRS_DESTBLEND,
DBLEND_INVSRCALPHA);

   // 纹理混合
   // 阶段 0:
   // ColorOp : 选择纹理
   // AlphaOp : 当前
   // 从 pTextureOut 的(tu1, tv1)为每一顶点查找 RGB 值。
```

```
   pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
0 );
   pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
   pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_TEXTURE );
   // 不设阿尔法值。
   pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_SELECTARG1);
   pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARG1,
D3DTA_CURRENT );

   // 阶段 1:
   // ColorOp : 当前
   // AlphaOp : 选择纹理
   RTN_HR_IF_FAILED(pd3dDevice->SetTexture(1,
m_pAlphaMapTexture));
   // RGB 值未从之前阶段改变
   pd3dDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX ,
1 );
   pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
   pd3dDevice->SetTextureStageState( 1, D3DTSS_COLORARG1,
D3DTA_CURRENT );
   // 从 pAlphaMapTexture 的(tu2, tv2)为每一顶点查找阿尔法值。
   pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
D3DTOP_SELECTARG1);
   pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAARG1,
D3DTA_TEXTURE );

   // 阶段 2: 禁用
   pd3dDevice->SetTextureStageState( 2, D3DTSS_ALPHAOP,
D3DTOP_DISABLE);
   pd3dDevice->SetTextureStageState( 2, D3DTSS_COLOROP,
```

```
D3DTOP_DISABLE );

   pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER,
D3DTEXF_LINEAR );
   pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR);

   pd3dDevice->SetSamplerState( 1, D3DSAMP_MINFILTER,
D3DTEXF_LINEAR);
   pd3dDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR);

   pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP);
   pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );

   // 将过渡模式网格的每一顶点的 tv2 设置为进度
   VERTEX_XYZ_TEX2* pVertices = NULL;
   pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pVertices );

   DWORD dwNumPatternVertices = pSysMemMesh->GetNumVertices();
   for(DWORD i = 0; i < dwNumPatternVertices; i++)
            pVertices[i].t2.y  = fltProgress; // 进度
   // 基于过渡持续时间、帧速率和当前帧计算 fltProgress。

   pSysMemMesh->UnlockVertexBuffer();
   m_pMesh->InvalidateDeviceObjects();
   m_pMesh->RestoreDeviceObjects(pd3dDevice);
   // RestoreDeviceObjects()将创建局部网格

   // 使用纹理 1 和过渡模式网格
   pd3dDevice->SetTexture(0, pTextureOut);
```

```
// pTextureOut 是存储外出视频帧的纹理对象。
// 将该纹理置为纹理混合阶段 0。

LPD3DXMESH pPatternMesh = m_pPatternMesh->GetLocalMesh();
pPatternMesh->DrawSubset( 0 ); // 使 D3D/GPU 执行使用该
```

PatternMesh、TextureOut 和当前设定的绘制。

为每一帧重复上述例程。当 D3D / GPU 绘制第一网格/进入视频源时，它将整个 TextureIn 置于呈现对象表面。当 D3D / GPU 绘制过渡模式网格/外出视频源时，它将从阿尔法映射中查找阿尔法值，将其与来自 TextureOut 的 RGB 值一起用于阿尔法混合计算，且 RGB 值在呈现对象中。

第 5 节：阿尔法映射生成示例代码。

每一顶点包括位置和颜色信息。

右上角(1, 0)是不透明的。右下角(0, 1) 是透明的。自左上至右下的线是 50%透明的。从透明到不透明的斜率遵循正弦曲线。将该网格呈现给纹理，拥有将用于创建软边的阿尔法映射。燃边（Burn Edge）的阿尔法映射具有相同形状但是不同颜色/阿尔法值。右上角(1, 0) 和右下角(0, 1) 是透明的。 从左上到右下的线具有由燃边强度设置的透明度。

```
struct ALPHAMAPVERTEX
{
    D3DXVECTOR3 v;          // 位置
    D3DCOLOR    color;      // 颜色
    static const DWORD FVF;
};
const DWORD ALPHAMAPVERTEX::FVF = D3DFVF_XYZ | D3DFVF_DIFFUSE;
        m_wNumSegments = 36;
        m_dwNumVertice
                = 2 // 左上和右下
                + (m_wNumSegments + 1) * 3; // 3: 左一中一右
        m_dwNumIndices
                = 2 * 4
                + (m_wNumSegments - 1) * 6;
```

```
            m_dwNumIndices *= 3;
   D3DXCreateMeshFVF(
            m_dwNumIndices / 3, // 面的个数,
           m_dwNumVertices,    // 顶点的个数,
            D3DXMESH_SYSTEMMEM, // 保存在相同存储器中,
            ALPHAMAPVERTEX::FVF,//顶点格式,
            pd3dDevice,         // GPU 实例,
            &pSysMemMesh        //返回 LPD3DXMESH
             );
   // 为写入锁定 VertexBuffer。
   ALPHAMAPVERTEX* pVertices = NULL;
   pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pVertices);
            D3DCOLOR clrRight;
            D3DCOLOR clrMiddle;
            D3DCOLOR clrLeft;
            if(m_Type == SoftEdge)
            {
                     clrLeft = 0;
                     clrMiddle = 0x80808080;
                     clrRight = 0xffffffff;
                    }
            else if(m_Type == BurnEdge)
            {
                     clrLeft = 0;
                     clrMiddle = D3DCOLOR_COLORVALUE
   (m_fltBurnIntensity,m_fltBurnIntensity,m_fltBurnIntensity,
m_fltBurnIntensity);
                     clrRight = 0;
            }
            else
            {
                     clrLeft = 0xffffffff;
                     clrMiddle = 0;
```

```
            clrRight = 0xffffffff;
        }
        // 左下
        pVertices[0].v.x   = 0.0f;
        pVertices[0].v.y   = 1.0f;
        pVertices[0].v.z   = 0.0f;
        pVertices[0].color = clrLeft;

       // 右上(对软边 100%; 对燃边 0)
        pVertices[1].v.x   = 1.0f;
        pVertices[1].v.y   = 0.0f;
        pVertices[1].v.z   = 0.0f;
        pVertices[1].color = clrRight;

// 左上
        pVertices[2].v.x   = 0.0f;
        pVertices[2].v.y   = 0.0f;
        pVertices[2].v.z   = 0.0f;
        pVertices[2].color = clrLeft;
        pVertices[3].v.x   = 0.0f;
        pVertices[3].v.y   = 0.0f;
        pVertices[3].v.z   = 0.0f;
        pVertices[3].color = clrMiddle;

        pVertices[4].v.x   = 0.0f;
        pVertices[4].v.y   = 0.0f;
        pVertices[4].v.z   = 0.0f;
        pVertices[4].color = clrRight;

        // 右下
        pVertices[5].v.x   = 1.0f;
        pVertices[5].v.y   = 1.0f;
        pVertices[5].v.z   = 0.0f;
        pVertices[5].color = clrLeft;
```

```
            pVertices[6].v.x   = 1.0f;
            pVertices[6].v.y   = 1.0f;
            pVertices[6].v.z   = 0.0f;
            pVertices[6].color = clrMiddle;

            pVertices[7].v.x   = 1.0f;
            pVertices[7].v.y   = 1.0f;
            pVertices[7].v.z   = 0.0f;
            pVertices[7].color = clrRight;

        WORD index = 8; // 0 到 7 已经在以上分配。

          // 使用从 0 到 180 度的正弦曲线；正弦值为 0 到 1 到 0。
        WORD i;
              for(i = 1; i < m_wNumSegments; i ++)
              {
                   FLOAT fltXY = (FLOAT)i /
(FLOAT)m_wNumSegments;
                   FLOAT fltAngle = D3DX_PI * fltXY;
                   FLOAT fltDistance = ::sin(fltAngle);
                   fltDistance *= m_fltGradientWidth / 2.0f;
                    // 调整渐变区域的宽度。
                   //检查 fltDistance 范围
                   if(fltXY < 0.5f)
                   // 上半部，所以 fltXY - fltDistance > 0
                         fltDistance = min(fltXY, fltDistance);
                   else if(fltXY > 0.5f)
                   // 下半部，所以 fltXY + fltDistance < 1
                         fltDistance = min(1.0f - fltXY,
fltDistance);
                   else
                   // 中间，所以 fltXY + fltDistance < 1 且 fltXY-
fltDistance > 0
                        fltDistance = min(0.5f, fltDistance);
```

```
                    // 左
                    pVertices[index].v.x   = fltXY - fltDistance;
                    pVertices[index].v.y   = fltXY;
                    pVertices[index].v.z   = 0.0f;
                    pVertices[index].color = clrLeft;

                     // 中
                    pVertices[index + 1].v.x   = fltXY;
                    pVertices[index + 1].v.y   = fltXY;
                    pVertices[index + 1].v.z   = 0.0f;
                    pVertices[index + 1].color = clrMiddle;

                     // 右
                    pVertices[index + 2].v.x   = fltXY +
fltDistance;
                    pVertices[index + 2].v.y   = fltXY;
                    pVertices[index + 2].v.z   = 0.0f;
                    pVertices[index + 2].color = clrRight;

                     index += 3;
                }

RTN_HR_IF_FAILED( pSysMemMesh->UnlockVertexBuffer());
                // 为写入锁定 IndexBuffer。
                WORD* pIndices = NULL;

RTN_HR_IF_FAILED( pSysMemMesh->LockIndexBuffer(D3DLOCK_NOSYSL
OCK, (void**)&pIndices));
                 RTN_HR_IF_BADPTR(pIndices);
               // 填充索引
               // 最左侧
               pIndices[0] = 2;
               pIndices[1] = 8;
               pIndices[2] = 0;
```

```
    // 左侧中心
    pIndices[3] = 3;
    pIndices[4] = 9;
    pIndices[5] = 8;

    // 右侧中心
    pIndices[6] = 3;
    pIndices[7] = 10;
    pIndices[8] = 9;

// 最后绘制最顶端，以确保顶端扫描线是完全不透明的。
index = 9;
for(i = 1; i < m_wNumSegments - 1; i ++)
 {
        // 至左
        pIndices[index + 0] = 0;
        pIndices[index + 1] = i * 3 + 5;
        pIndices[index + 2] = i * 3 + 8;

         // 左象限
          pIndices[index + 3] = i * 3 + 5;
    pIndices[index + 4] = i * 3 + 6;
    pIndices[index + 5] = i * 3 + 8;

    pIndices[index + 6] = i * 3 + 6;
    pIndices[index + 7] = i * 3 + 9;
    pIndices[index + 8] = i * 3 + 8;

    // 右象限
       pIndices[index + 9] = i * 3 + 6;
    pIndices[index + 10] = i * 3 + 7;
    pIndices[index + 11] = i * 3 + 9;
```

```
            pIndices[index + 12] = i * 3 + 7;
            pIndices[index + 13] = i * 3 + 10;
            pIndices[index + 14] = i * 3 + 9;

            // 至右
            pIndices[index + 15] = 1;
            pIndices[index + 16] = i * 3 + 10;
            pIndices[index + 17] = i * 3 + 7;

            index += 18;
        }
        // 最右
        pIndices[index + 0] = 7;
        pIndices[index + 1] = i * 3 + 7;
        pIndices[index + 2] = 1;

        // 右侧中心
        pIndices[index + 3] = 6;
        pIndices[index + 4] = i * 3 + 6;
        pIndices[index + 5] = i * 3 + 7;

        // 左侧中心
        pIndices[index + 6] = 6;
        pIndices[index + 7] = i * 3 + 5;
        pIndices[index + 8] = i * 3 + 6;

        // 最底部
        pIndices[index + 9] = 5;
        pIndices[index + 10] = 0;
        pIndices[index + 11] = i * 3 + 5;

        // 最顶端
        pIndices[index + 12] = 4;
        pIndices[index + 13] = 1;
```

```
        pIndices[index + 14] = 10;
    }
    RTN_HR_IF_FAILED( pSysMemMesh->UnlockIndexBuffer());
      // 创建CDXUTMesh对象以在系统存储器和设备存储器中管理D3DXMesh对
    象。
    m_pMesh = new CDXUTMesh( L"CAlphaMap" );
       RTN_HR_IF_BADNEW(m_pMesh);

    m_pMesh->m_pSysMemMesh = pSysMemMesh.Detach();
    //使之前创建的D3DXMesh对象受控于CDXUTMesh。
       m_pMesh->RestoreDeviceObjects(pd3dDevice);
       // 在设备存储器中创建副本。

    #ifdef _DEBUG
    // 网格可被保存至.x文件。
       if(get_MeshFilename( ))
       {
          VERIFY_OK(SaveMesh(pd3dDevice, get_MeshFilename( )));
       }
    #endif
    // 使用128 x 256阿尔法映射
    // 确保第一扫描线是完全不透明的。
    // 且最后的扫描线是完全透明的。
    CComPtr<IDirect3DSurface9> pSurfaceLevel;
       pd3dDevice->CreateRenderTarget(
          128, 256,           //UINT宽度, UINT高度,
          D3DFMT_A8R8G8B8,     //呈现目标格式,
          D3DMULTISAMPLE_NONE, //D3DMULTISAMPLE_TYPE,
          D3DMULTISAMPLE_NONE, //多样值质量,
          TRUE,               //可查找BOOL,
          &pSurfaceLevel,      // 返回IDirect3DSurface9**
          ppSurface,
          NULL                //HANDLE* pSharedHandle
```

```
    );
    // 使D3D/GPU呈现给该表面。
    pd3dDevice->SetRenderTarget(0, pSurfaceLevel);

  // 设置D3D状态。
  pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
    pd3dDevice->SetRenderState( D3DRS_SRCBLEND,
  D3DBLEND_SRCALPHA );
    pd3dDevice->SetRenderState( D3DRS_DESTBLEND,
  D3DBLEND_ZERO ); // 无需混合现有像素。

    pd3dDevice->SetRenderState(D3DRS_SHADEMODE,
  D3DSHADE_GOURAUD); //平滑着色
  pd3dDevice->SetRenderState(D3DRS_COLORWRITEENABLE,
        D3DCOLORWRITEENABLE_RED  | D3DCOLORWRITEENABLE_GREEN
        | D3DCOLORWRITEENABLE_BLUE |
        D3DCOLORWRITEENABLE_ALPHA );

  D3DXMATRIX matWorld;
    D3DXMatrixIdentity( &matWorld );
  pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld ); // 无缩
放/平移/旋转。

  // 设置D3D照相机。
  D3DXMATRIX matView;
  D3DXVECTOR3 vFromPt   = D3DXVECTOR3( 0.5f, 0.5f, 10.0f );
  D3DXVECTOR3 vLookatPt = D3DXVECTOR3( 0.5f, 0.5f, 0.0f );
  D3DXVECTOR3 vUpVec    = D3DXVECTOR3( 0.0f, -1.0f, 0.0f );
  D3DXMatrixLookAtLH( &matView, &vFromPt, &vLookatPt, &vUpVec );
  pd3dDevice->SetTransform( D3DTS_VIEW, &matView );

  D3DXMATRIX matProj;
  D3DXMatrixOrthoLH( &matProj, 1.0f, 1.0f, 0.01f, 100.0f );
  pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
```

```
    //设置D3D纹理混合阶段状态
   pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
   D3DTOP_SELECTARG1 );
     pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
  D3DTA_DIFFUSE);
   pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
   D3DTOP_SELECTARG1 );
     pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARG1,
  D3DTA_DIFFUSE);

   pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
   D3DTOP_DISABLE );
   pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
   D3DTOP_DISABLE );

     RTN_HR_IF_BADNEW(m_pMesh);
  LPD3DXMESH pMesh = m_pMesh->GetLocalMesh();
     RTN_HR_IF_BADNEW(pMesh);

  // 清空视见区
  pd3dDevice->Clear( 0L, NULL, D3DCLEAR_TARGET, 0x00000000, 1.0f,
0L);

  // 开始场景
  pd3dDevice->BeginScene();
     HRESULT hr = S_OK;

     // 执行绘制
     pMesh->DrawSubset( 0 ));

  // 结束场景。
  pd3dDevice->EndScene();
```

```
// 创建纹理以存储阿尔法映射。
CComPtr<IDirect3DTexture9> pTexture;
   if(S_OK != pd3dDevice->CreateTexture(
      128, 256,   //UINT 宽度，UINT 高度，
      1,          //MIP等级，
      D3DUSAGE_DYNAMIC,
      D3DFMT_A8,  //纹理格式，仅阿尔法。
      D3DPOOL_DEFAULT, // 置于设备存储器中。
      &pTexture,  // 返回纹理
      NULL)       //HANDLE* pSharedHandle) ||
      || pTexture == NULL)
   {
      // 未能创建A8格式纹理
      // 尝试A8R8G8B8格式。使用更多存储器。

      pd3dDevice->CreateTexture(
         128, 256,
         1,
         D3DUSAGE_DYNAMIC,
         D3DFMT_A8R8G8B8,
         D3DPOOL_DEFAULT,
         &pTexture,
         NULL
      );
}
CComPtr<IDirect3DSurface9> pDestSurfaceLevel;
pTexture->GetSurfaceLevel(0/* First MIP Level*/,
&pDestSurfaceLevel);
// 复制RenderTarget像素至纹理。
D3DXLoadSurfaceFromSurface(
      pDestSurfaceLevel, //LPDIRECT3DSURFACE9 pDestSurface,
      NULL,              //CONST PALETTEENTRY *pDestPalette,
      NULL,              //CONST RECT *pDestRect,
      pSurfaceLevel,     //LPDIRECT3DSURFACE9 pSrcSurface,
```

```
        NULL,              //CONST PALETTEENTRY *pSrcPalette,
        NULL,              //CONST RECT *pSrcRect,
        D3DX_FILTER_POINT, //DWORD Filter,
        0           //D3DCOLOR ColorKey
    );
    *ppTexture = pTexture.Detach();
    // 该纹理是阿尔法映射。

    #ifdef _DEBUG
    // 阿尔法映射可被保存至.dds或jpg文件。
       D3DXSaveSurfaceToFile(
       _T(".\\Shared\\RuntimeAlphaMap.dds"),   //文件名
       D3DXIFF_DDS,                            //文件格式,
       pDestSurfaceLevel,                 //将该表面保存至文件,
       NULL,                                   //无PALETTEENTRY,
       NULL                               //无RECT; 整个表面
    );
    #endif
```

第 6 节：燃边示例代码。

// 当使用锐边或软边技术呈现擦除过渡之后，使用燃边阿尔法映射将花边纹理呈现给过渡模式网格。燃边阿尔法映射具有与软边技术的阿尔法映射相同的形状顶点安排。但是颜色安排不同。 它在左上到右下的线的两边都是透明的。左上至右下线的阿尔法值是 BurnIntensity。从该中心线，阿尔法值逐渐降至 0。

```
    // 设置呈现状态
    pd3dDevice->SetRenderState( D3DRS_SHADEMODE,
D3DSHADE_GOURAUD);
    pd3dDevice->SetRenderState( D3DRS_ZENABLE, D3DZB_FALSE);
    pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE, FALSE);
    pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE, FALSE);
    pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, FALSE);
    pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
```

```
   pd3dDevice->SetRenderState( D3DRS_SRCBLEND,D3DBLEND_SRCALP
HA );
   pd3dDevice->SetRenderState(D3DRS_DESTBLEND,D3DBLEND_INVSRC
ALPHA);
   if(m_enumFancyEdge != FancyEdgeColor && m_enumFancyEdge !=
FancyEdgeColors)
   {
      pd3dDevice->SetRenderState( D3DRS_DESTBLEND,
D3DBLEND_ONE );
      // 所以花边纹理中的黑色像素将不显示。
   }
   // 纹理混合
   // 阶段 0:
   // ColorOp : 选择纹理
   // AlphaOp : 当前
   pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
0 );
   pd3dDevice->SetTextureStageState( 0,
D3DTSS_COLOROP,   D3DTOP_SELECTARG1 );
   pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_TEXTURE );
   if(m_enumFancyEdge == FancyEdgeColor || m_enumFancyEdge ==
FancyEdgeColors)
   {
       pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
     D3DTA_TFACTOR );
      //从 RenderState TextureFactor 获取 RGB 值。
   }
   pd3dDevice->SetTextureStageState( 0,
D3DTSS_ALPHAOP,   D3DTOP_SELECTARG1 );
   pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARG1,
D3DTA_CURRENT );

   // 阶段 1:
```

```
    // ColorOp : 当前
    // AlphaOp : 选择纹理
    pd3dDevice->SetTexture(1, m_pAlphaMapTexture);
    pd3dDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX ,
1 );
    pd3dDevice->SetTextureStageState( 1,
D3DTSS_COLOROP,   D3DTOP_SELECTARG1 );
    pd3dDevice->SetTextureStageState( 1, D3DTSS_COLORARG1,
D3DTA_CURRENT );
    pd3dDevice->SetTextureStageState( 1,
D3DTSS_ALPHAOP,   D3DTOP_SELECTARG1 );
    pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAARG1,
D3DTA_TEXTURE );

    // 阶段 2：禁用
    pd3dDevice->SetTextureStageState( 2,
D3DTSS_ALPHAOP, D3DTOP_DISABLE );
    pd3dDevice->SetTextureStageState( 2,
D3DTSS_COLOROP,   D3DTOP_DISABLE );

    pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER,
D3DTEXF_LINEAR );
    pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR );

    pd3dDevice->SetSamplerState( 1, D3DSAMP_MINFILTER,
D3DTEXF_LINEAR );
    pd3dDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR );

    pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP );
    pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );
```

```
   if(m_enumFancyEdge == FancyEdgeColor || m_enumFancyEdge ==
FancyEdgeColors)
   {
      if(m_enumFancyEdge == FancyEdgeColors)
      {
         D3DXCOLOR clrStart( m_colorFancyEdgeStart );
         D3DXCOLOR clrStop( m_colorFancyEdgeStop );
         D3DXCOLOR clrOut( (DWORD)0 );

            D3DXColorLerp(&clrOut, &clrStart, &clrStop,
          (FLOAT)fTime);

            D3DCOLOR  clr =
          D3DCOLOR_COLORVALUE(clrOut.r,clrOut.g,clrOut.b,clrOu
          t.a);
         pd3dDevice->SetRenderState( D3DRS_TEXTUREFACTOR, clr );
      }
      else
      {
            pd3dDevice->SetRenderState( D3DRS_TEXTUREFACTOR,
          m_colorFancyEdgeStart );
      }
   }
   else
   {
      pd3dDevice->SetTexture(0, m_pFancyEdgeTexture);
   }

   pPatternMesh->DrawSubset( 0 ); //使 D3D/GPU 执行使用该
PatternMesh、TextureOut 以及当前设定的绘制。
```

第 7 节：花边示例代码。

```
   FLOAT fltShift = m_fltGradientWidth;
   if(m_fltGradientWidth == 0.0f)
   {
```

```
        fltShift = 0.05f;
    }

    fltProgress = (FLOAT)fTime;

    if(m_fProgressShift)
    {
        if(fltProgress > fltShift)
        {
            fltProgress = fltShift + fltProgress;
        }
        else
        {
            fltProgress *= 2.0f;
        }
    }
```

在一个实施例中，通过与锐边/软边/消隐边技术完全相同的进程呈现过渡模式网格两次。第一遍呈现可以是火焰图像或波浪图像的花边纹理。第二遍呈现外出视频源以及按上述源代码移位的过程。

第 8 节：消隐边示例代码。

```
// 首先绘制进入视频源。

// 设置Direct3D状态

// pd3dDevice是Direct3D / GPU的实例指针。

pd3dDevice->SetRenderState(D3DRS_ALPHATESTENABLE, FALSE); //
无需执行阿尔法测试。

pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,FALSE);
// 无需执行阿尔法混合。
```

```
    pd3dDevice->SetRenderState( D3DRS_STENCILENABLE,    FALSE);
// 无需执行模板测试.
    pd3dDevice->SetRenderState( D3DRS_ZENABLE,
D3DZB_FALSE); // 无需执行深度测试
    pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE,     FALSE);
// 无需在深度缓冲器中保存Z值

    // 设置Direct3D纹理混合状态。
    // Direct3D允许至多8个阶段，但仅需要1个。

    //从pTextureIn的(tu1, tv1)为每一顶点查找RGB值。
    pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
0 ); // 为阶段0使用t0纹理坐标。
    pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARG1); // 输出RGB值

    pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_TEXTURE );// 从纹理中输出RGB值
    pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
    // 没有从阶段0及以上的阿尔法输出
    pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_DISABLE );

    // 没有从阶段1及以上的RGB输出。
    // 绘制进入源
    // 使用纹理2和第一网格

    pd3dDevice->SetTexture(0, pTextureIn);
    // pTextureIn 是存储进入视频帧的纹理对象。
    // 将该纹理置为纹理混合阶段0。

    LPD3DXMESH pPlaneMesh = m_pPlaneMesh->GetLocalMesh(); // 从
```

CDXUTMesh获取存储在设备/GPU存储器中的D3DXMesh对象

pPlaneMesh->DrawSubset(0); // 使D3D/GPU执行使用该PlaneMesh、TextureIn和当前设定的绘制

// 绘制外出源

// 将过渡模式网格的每一顶点的tv2设置成进度

```
VERTEX_XYZ_TEX2* pVertices = NULL;
pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pVertices );

DWORD dwNumPatternVertices = pSysMemMesh->GetNumVertices();
for(DWORD i = 0; i < dwNumPatternVertices; i++)
pVertices[i].t2.y  = fltProgress; // 进度
```

// 基于过渡持续时间、帧速率和当前帧计算fltProgress。

```
pSysMemMesh->UnlockVertexBuffer();
m_pMesh->InvalidateDeviceObjects();
m_pMesh->RestoreDeviceObjects(pd3dDevice);
```

// RestoreDeviceObjects()将创建局部网格

//消隐特效技术需要呈现模式网格两次。第一遍，它将来自消隐映射和阿尔法映射的阿尔法值相加。该值与阿尔法测试操作中的预定义阈值进行比较。为通过测试的像素增加模板值。RGB值不写入呈现对象。

//使用src=zero且dest=one的阿尔法混合以防止写入颜色缓冲器

```
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE);
pd3dDevice->SetRenderState( D3DRS_SRCBLEND,
D3DBLEND_ZERO);
```

```
    pd3dDevice->SetRenderState( D3DRS_DESTBLEND,
D3DBLEND_ONE);

    // 禁用深度缓冲
    pd3dDevice->SetRenderState( D3DRS_ZENABLE,
D3DZB_FALSE);
    pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE,  FALSE);

    //设置模板函数以增加通过阿尔法测试的像素的模板
    pd3dDevice->SetRenderState( D3DRS_STENCILPASS,
D3DSTENCILOP_INCR);
    pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, TRUE);
    pd3dDevice->SetRenderState( D3DRS_STENCILFUNC,
D3DCMP_ALWAYS);

    //设置阿尔法测试
    pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE,TRUE);
    pd3dDevice->SetRenderState( D3DRS_ALPHAFUNC,D3DCMP_GREATER
);
    pd3dDevice->SetRenderState( D3DRS_ALPHAREF, 0x40);

    // 纹理混合
    // 阶段0:
    // ColorOp : 当前
    // AlphaOp : 选择纹理
    RTN_HR_IF_FAILED(pd3dDevice->SetTexture(0,
m_pAlphaMapTexture));
    pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
1 );
    pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
    pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_CURRENT );
    pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
```

```
D3DTOP_SELECTARG1
    );
    pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARG1,
D3DTA_TEXTURE );

    // 阶段1:
    // ColorOp : 当前
    // AlphaOp : 带符号加法
    RTN_HR_IF_FAILED(pd3dDevice->SetTexture(1,
m_pDissolveMapTexture));
    pd3dDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX ,
0 );
    pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
    pd3dDevice->SetTextureStageState( 1, D3DTSS_COLORARG1,
D3DTA_CURRENT );
    // 将来自两个阿尔法纹理的阿尔法值相加。
    pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
D3DTOP_ADDSIGNED);
    pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAARG1,
D3DTA_TEXTURE );

    // 阶段2: 禁用
    pd3dDevice->SetTextureStageState( 2, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
    pd3dDevice->SetTextureStageState( 2, D3DTSS_COLOROP,
D3DTOP_DISABLE );

    pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER,
D3DTEXF_LINEAR );
    pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR );

    pd3dDevice->SetSamplerState( 1, D3DSAMP_MINFILTER,
```

```
D3DTEXF_POINT );
   pd3dDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER,
D3DTEXF_POINT );

   pd3dDevice->SetSamplerState(0, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP );
   pd3dDevice->SetSamplerState(0, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );

   pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP );
   pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );

   LPD3DXMESH pPatternMesh = m_pPatternMesh->GetLocalMesh();
   pPatternMesh->DrawSubset( 0 ); // 使D3D/GPU执行使用PatternMesh
以及当前设定的绘制。

   // 第二遍，它将执行深度测试和模板测试。
   // 只有通过这两项测试的像素被绘制到呈现对象中。
   // 模板测试形成沿擦除边的随机模式。
   // 深度测试是为确保没有像素在边的错误一侧出现。

   // 为呈现视频1设置呈现状态
   // 禁用阿尔法混合与测试。
   pd3dDevice->SetRenderState(D3DRS_ALPHATESTENABLE,   FALSE);
   pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,
FALSE);

   // 为呈现第一场景设置模板状态。
   pd3dDevice->SetRenderState( D3DRS_STENCILENABLE,   TRUE);
   pd3dDevice->SetRenderState( D3DRS_STENCILPASS,
D3DSTENCILOP_KEEP);
   pd3dDevice->SetRenderState( D3DRS_STENCILREF,      1);
```

```
   pd3dDevice->SetRenderState( D3DRS_STENCILFUNC,
D3DCMP_EQUAL);

   pd3dDevice->SetRenderState( D3DRS_ZENABLE,     D3DZB_TRUE);
   pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE, TRUEE);

   pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX ,
0 );
   pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
   pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARG1,
D3DTA_TEXTURE );
   pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
   pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_DISABLE );

   pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER,
D3DTEXF_LINEAR );
   pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR );

   // 使用纹理1和过渡模式网格
   pd3dDevice->SetTexture(0, pTextureOut);
   // pTextureOut是存储外出视频帧的纹理对象。
   // 将该纹理置为纹理混合阶段0。

   // 应用平移变换矩阵使过渡模式网格移动到z方向上较低位置。

   D3DXMATRIX matTranslation;
   D3DXMatrixTranslation( &matTranslation, 0, 0, -fltProgress +
   m_fltGradientWidth);

   // 创建平移变换矩阵以将模式网格移动至z方向上较低位置。
```

```
pd3dDevice->SetTransform( D3DTS_WORLD, &matTranslation );

// 设置D3D世界变换矩阵。
// D3D世界变换将模型空间坐标转化成世界空间。
// 过渡模式网格中的(0, 0, 0)变成 (0, 0, -fltProgress +
m_fltGradientWidth)
pPatternMesh->DrawSubset( 0 ); // 使D3D/GPU执行使用PatternMesh、
TextureOut以及当前设定的绘制。
```

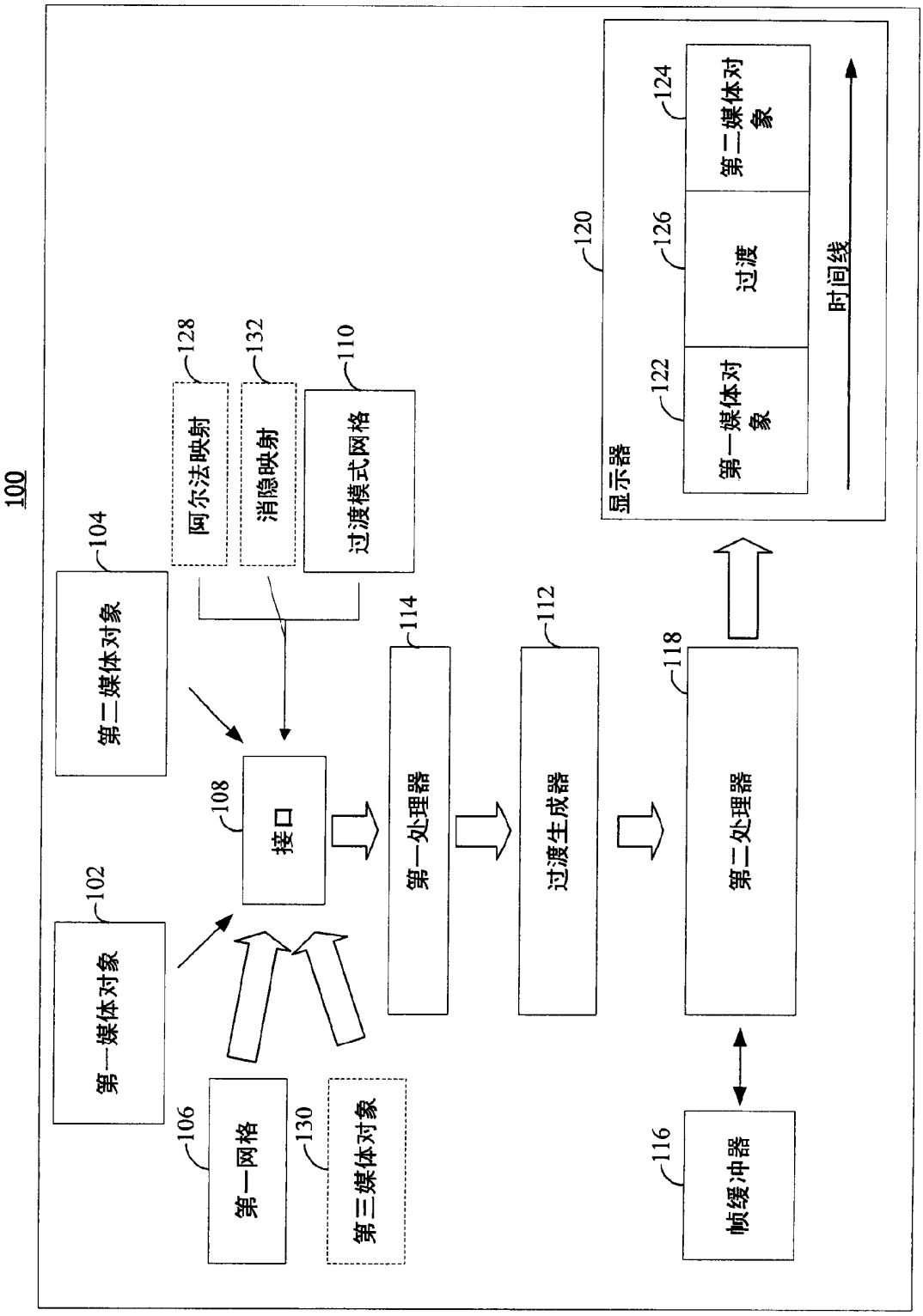

100
第一媒体对象
102
第二媒体对象
104
第一网格
106
接口
108
阿尔法映射
128
消隐映射
132
过渡模式网格
110
第三媒体对象
130
第一处理器
114
过渡生成器
112
第二处理器
118
帧缓冲器
116
显示器
120
第一媒体对象
122
过渡
126
第二媒体对象
124
时间线

图 1

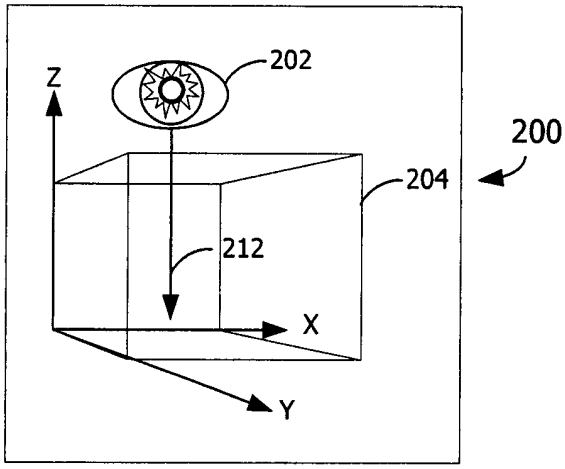


图 2A

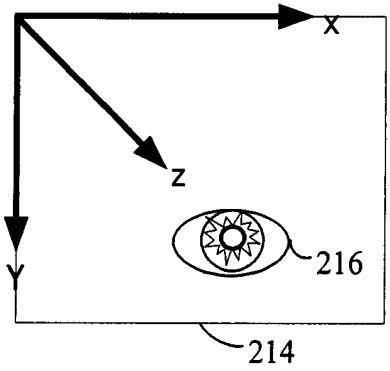
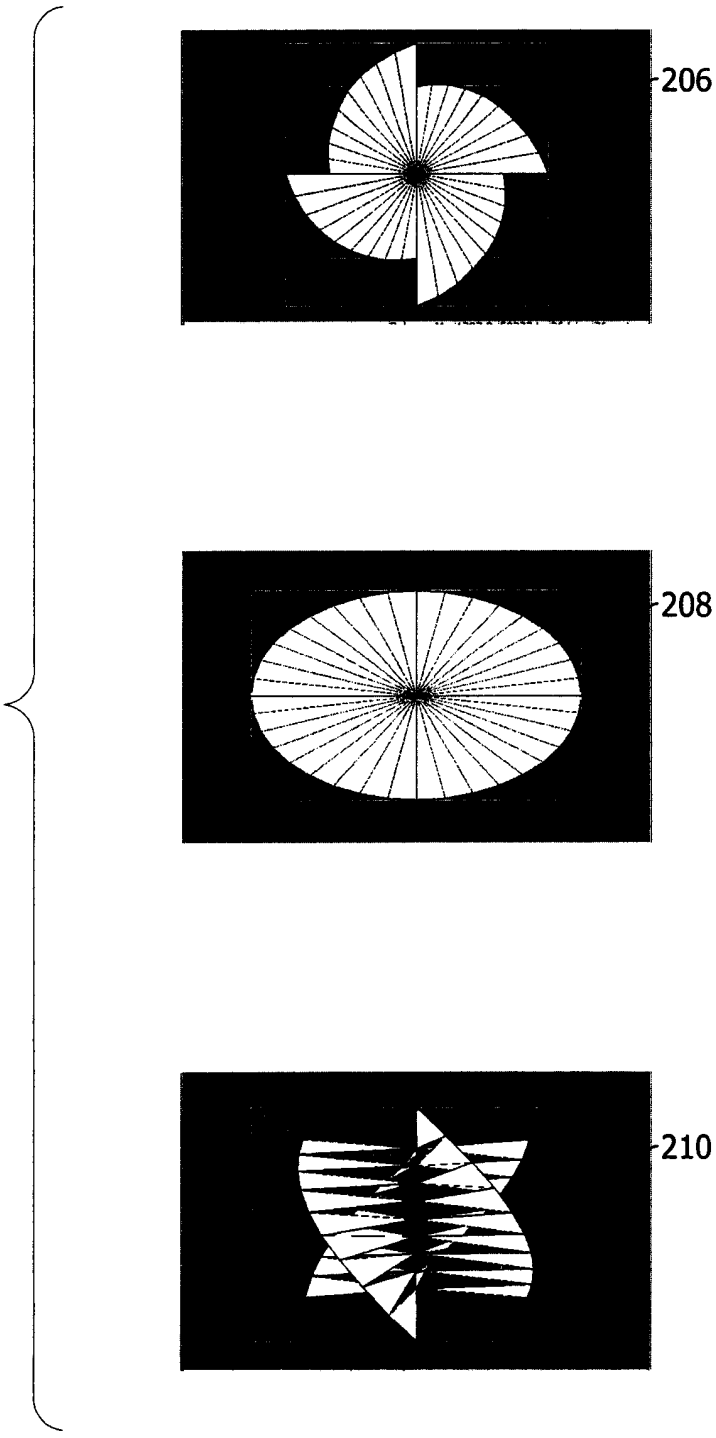


图 2B

图 2C



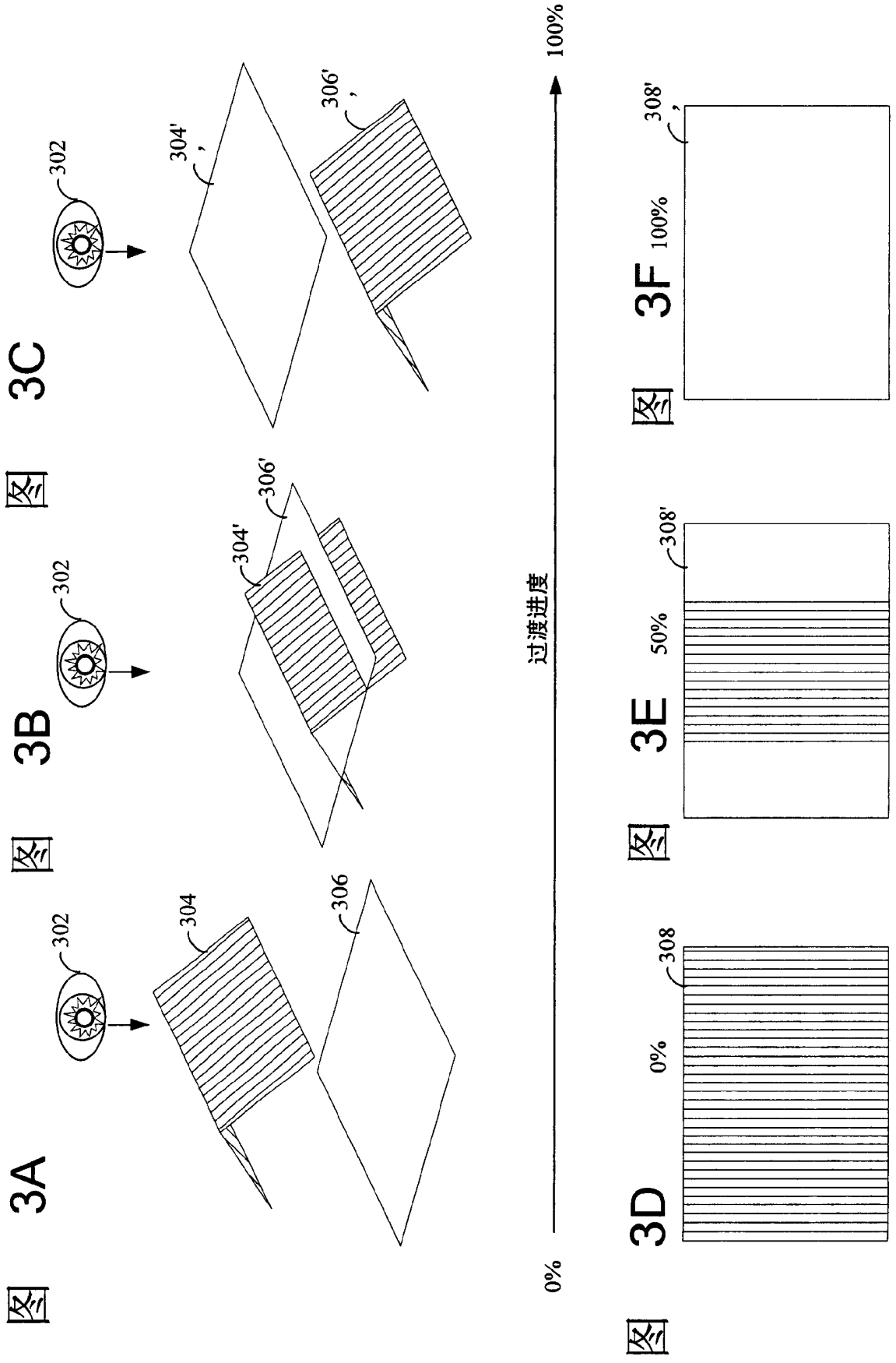


图 3A
302
304
306
图 3B
302
304'
306'
图 3C
302
304''
306''
过渡进度
0%
100%
图 3D
0%
308
图 3E
50%
308'
图 3F
100%
308''

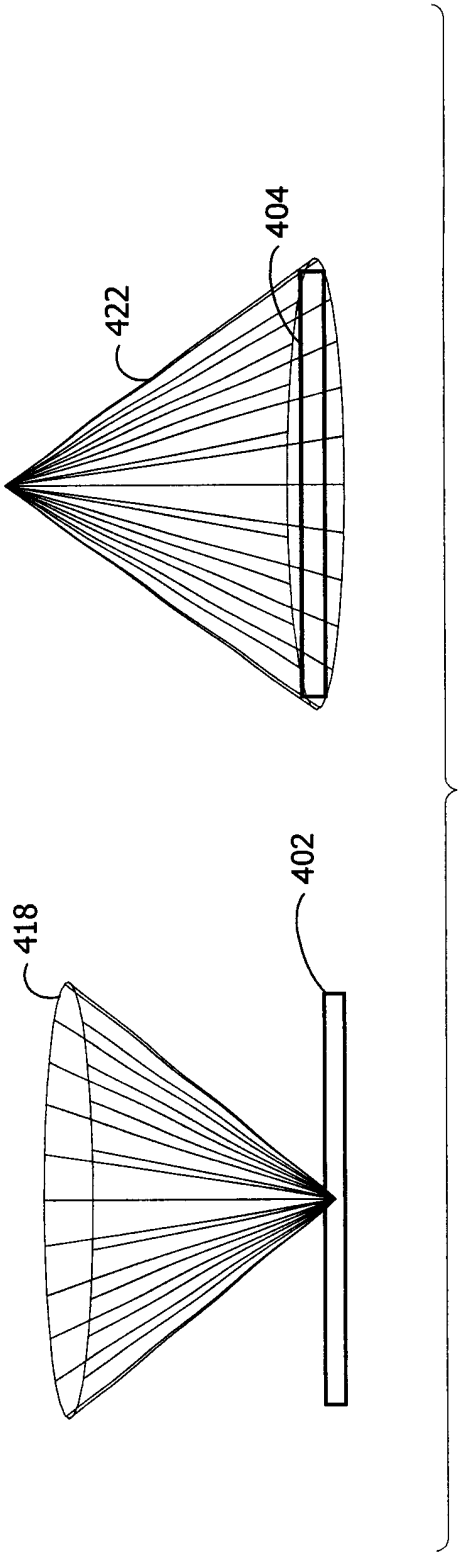


图 4A

图 4B

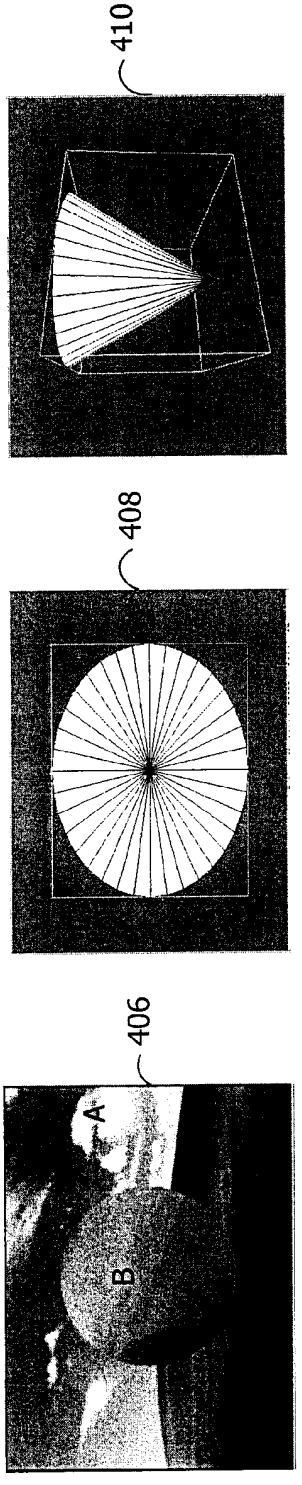


图 4C

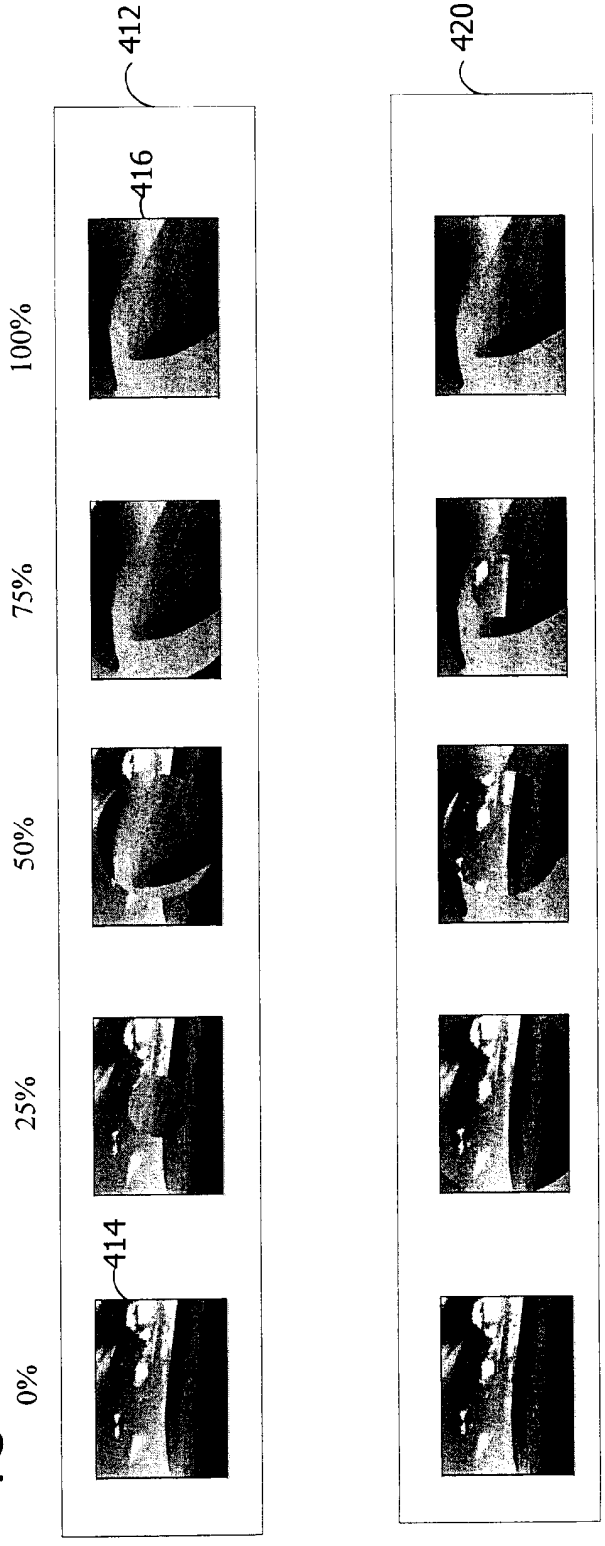


图 5A

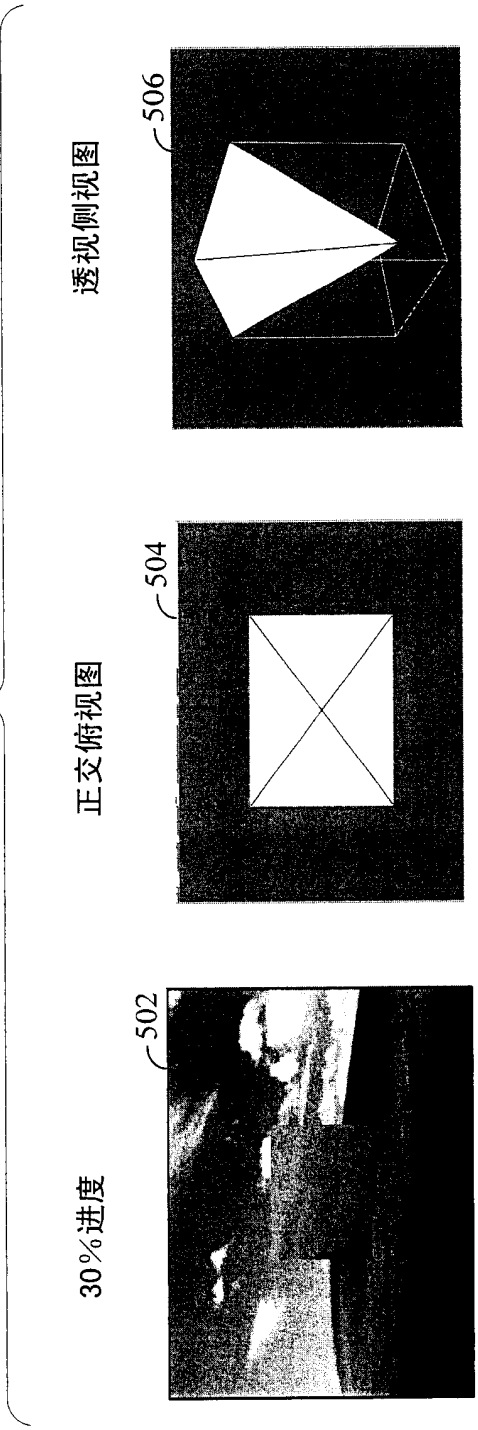
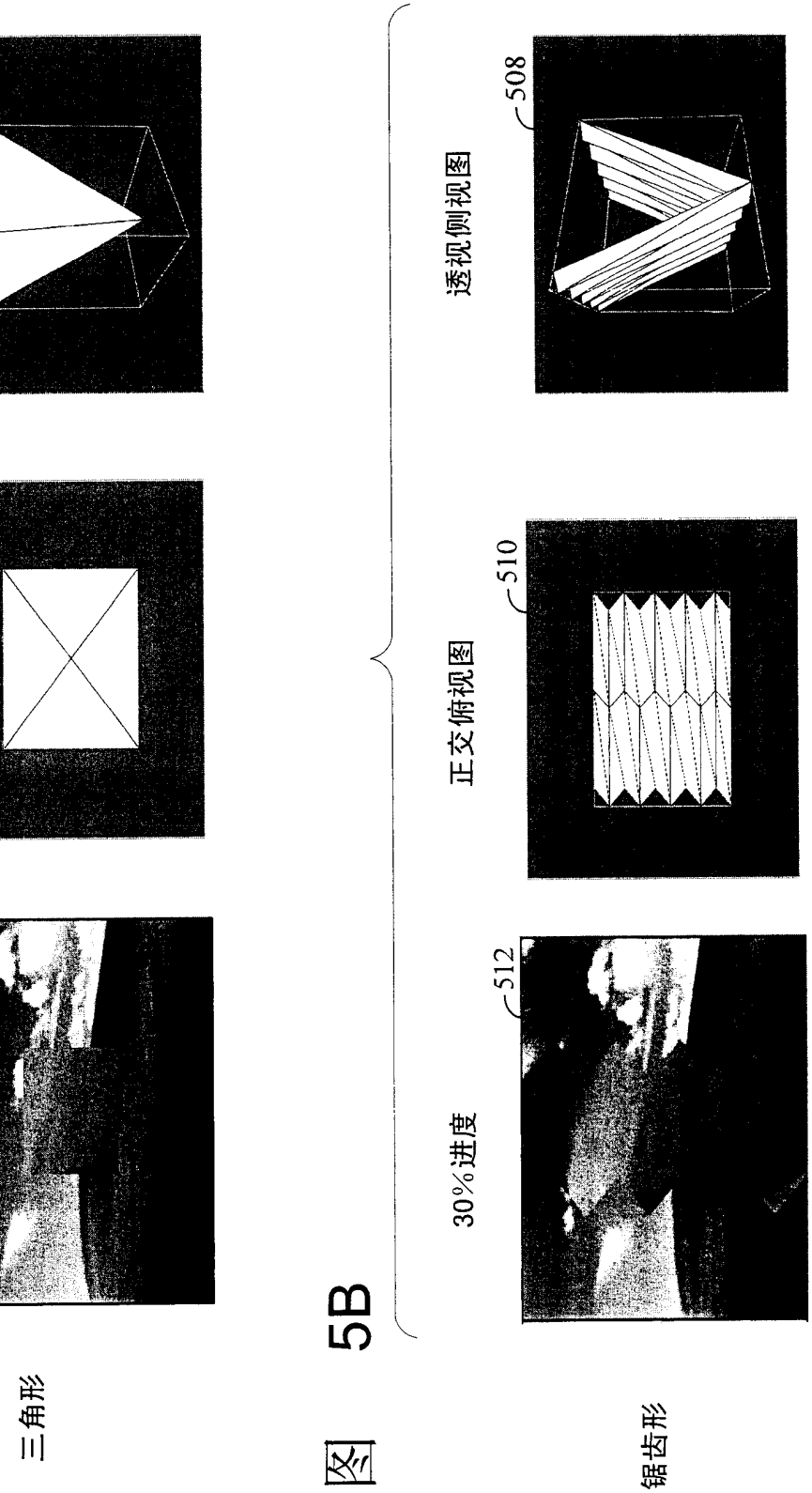

30%进度
502
正交俯视图
504
透视侧视图
506
三角形

图 5B


30%进度
512
正交俯视图
510
透视侧视图
508
锯齿形

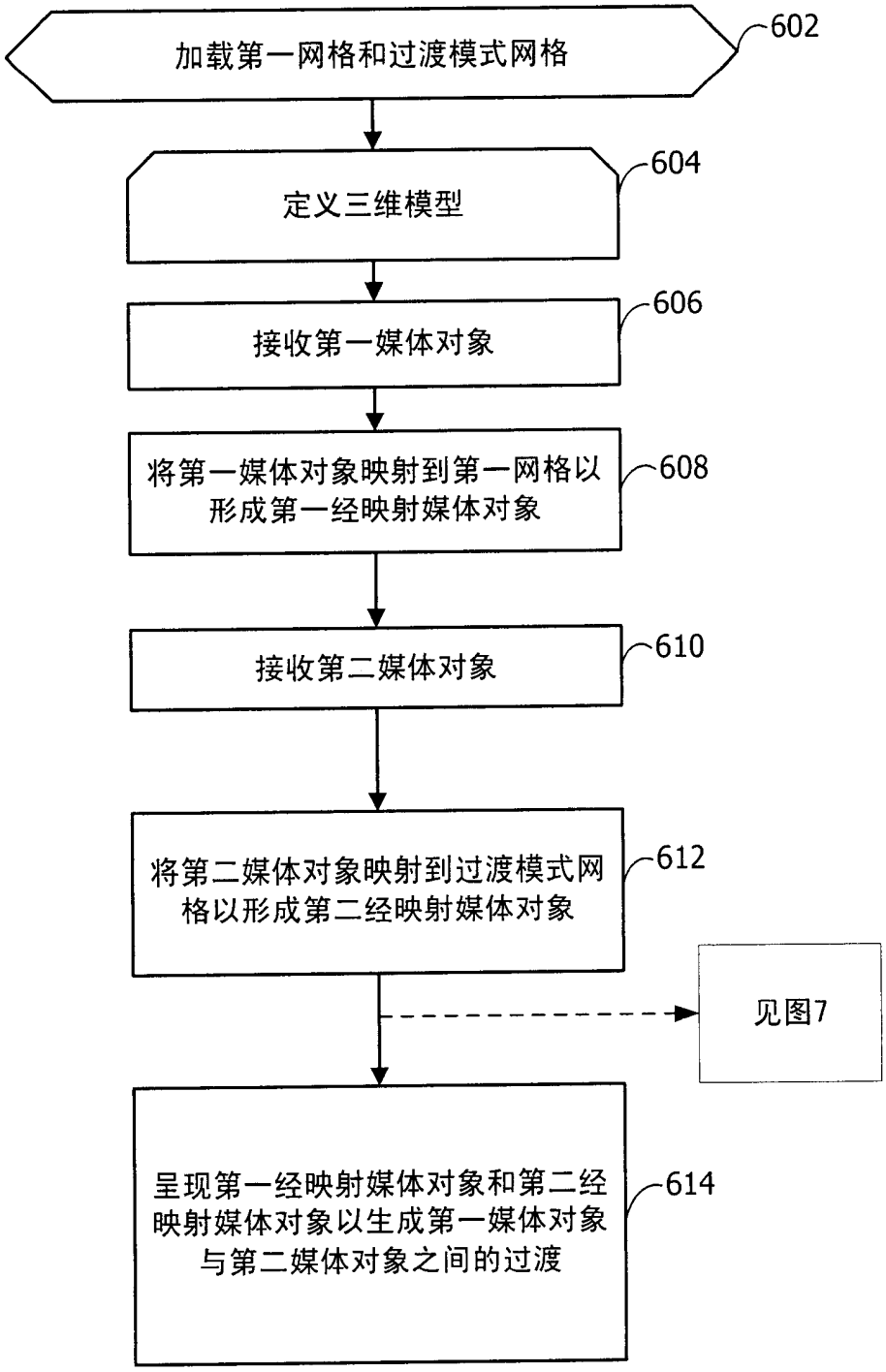

加载第一网格和过渡模式网格
602
定义三维模型
604
接收第一媒体对象
606
将第一媒体对象映射到第一网格以形成第一经映射媒体对象
608
接收第二媒体对象
610
将第二媒体对象映射到过渡模式网格以形成第二经映射媒体对象
612
见图7
呈现第一经映射媒体对象和第二经映射媒体对象以生成第一媒体对象与第二媒体对象之间的过渡
614

图 6

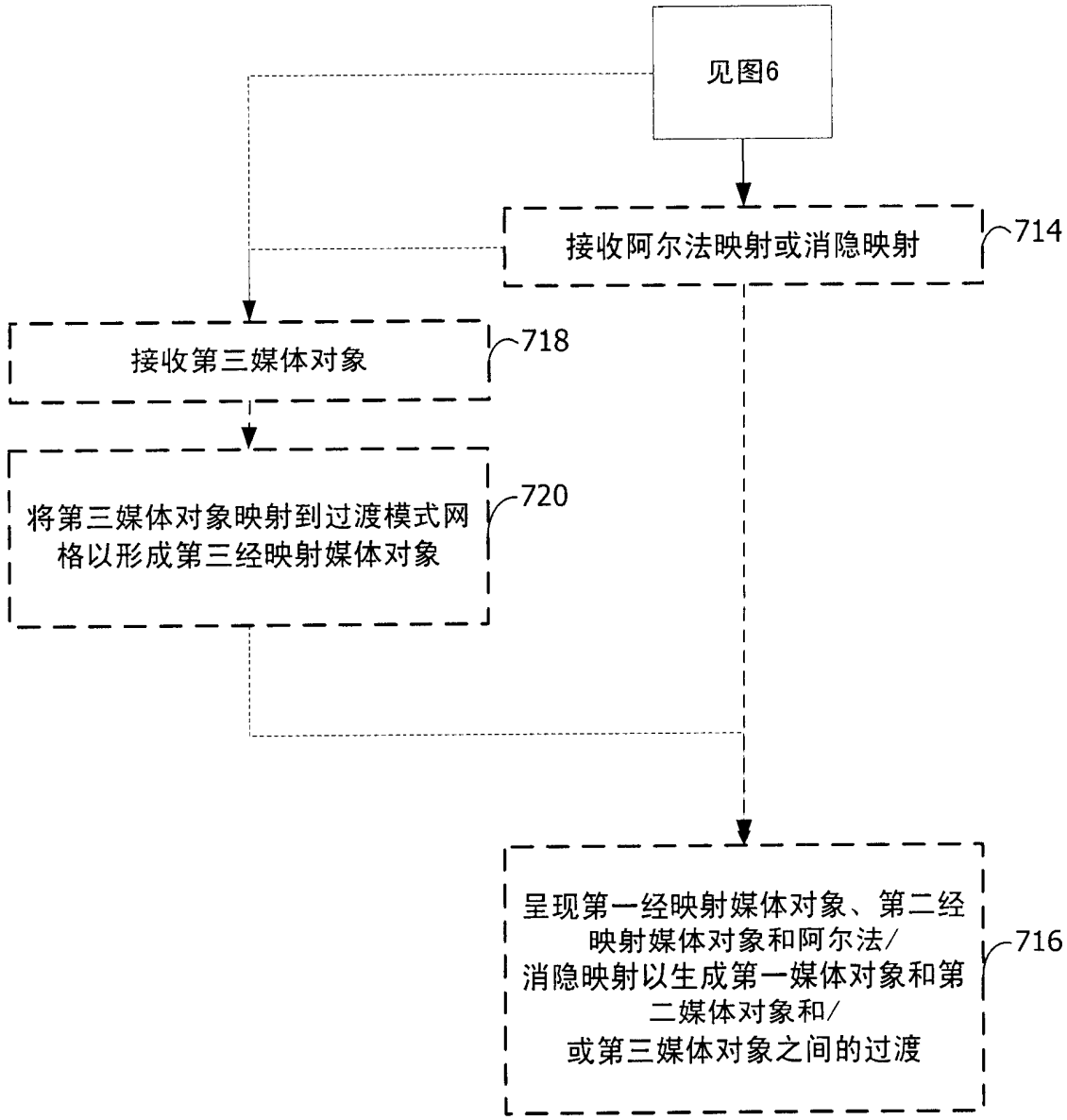
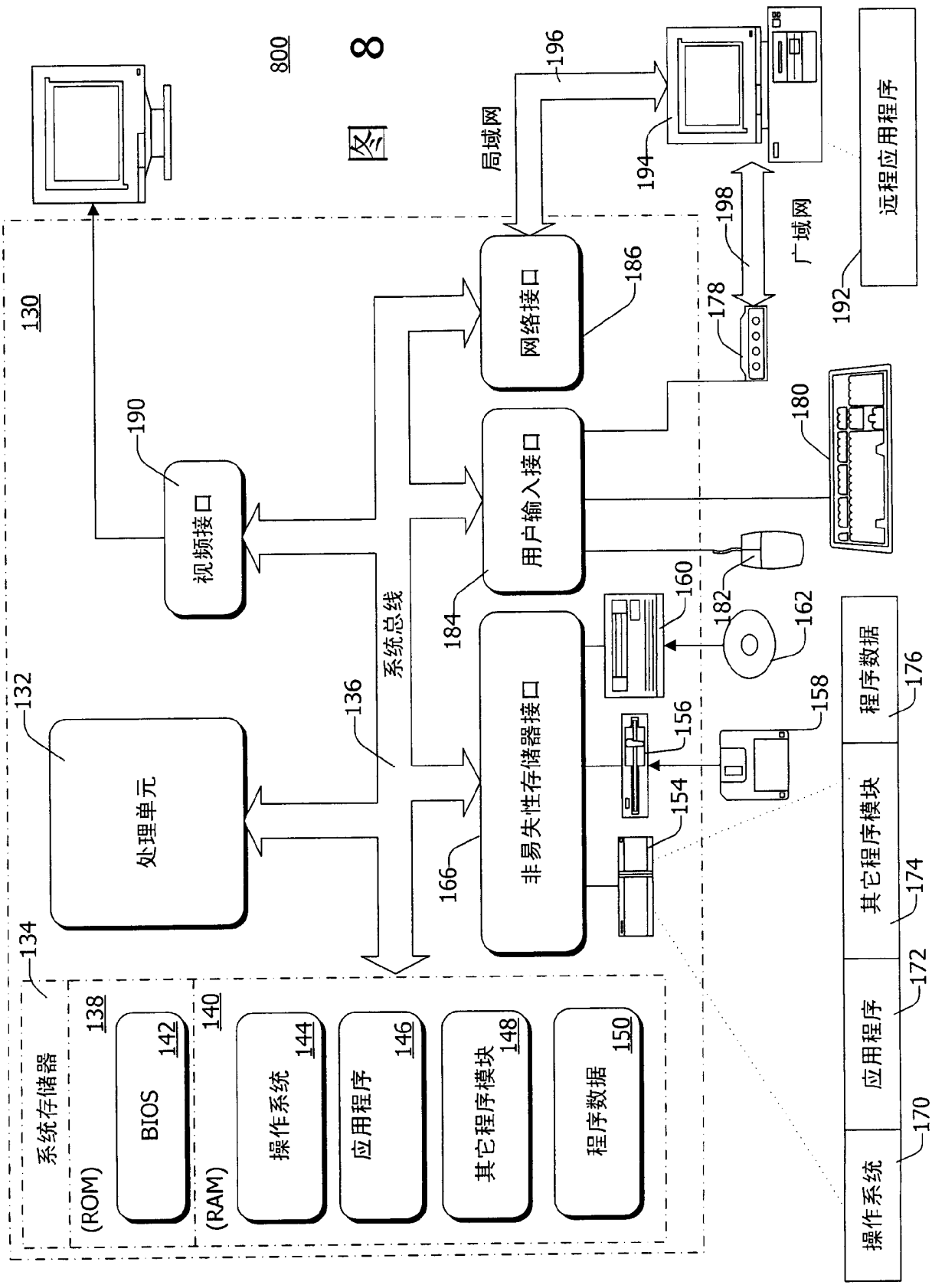

见图6
接收阿尔法映射或消隐映射
714
接收第三媒体对象
718
将第三媒体对象映射到过渡模式网格以形成第三经映射媒体对象
720
呈现第一经映射媒体对象、第二经映射媒体对象和阿尔法/消隐映射以生成第一媒体对象和第二媒体对象和/或第三媒体对象之间的过渡
716

图 7


800
图 8
130
132
处理单元
134
系统存储器
(ROM)
138
BIOS
142
(RAM)
140
操作系统
144
应用程序
146
其它程序模块
148
程序数据
150
136
系统总线
190
视频接口
166
非易失性存储器接口
184
用户输入接口
186
网络接口
160
162
154
156
158
182
180
178
198
广域网
局域网
196
194
192
远程应用程序
170
操作系统
172
应用程序
174
其它程序模块
176
程序数据

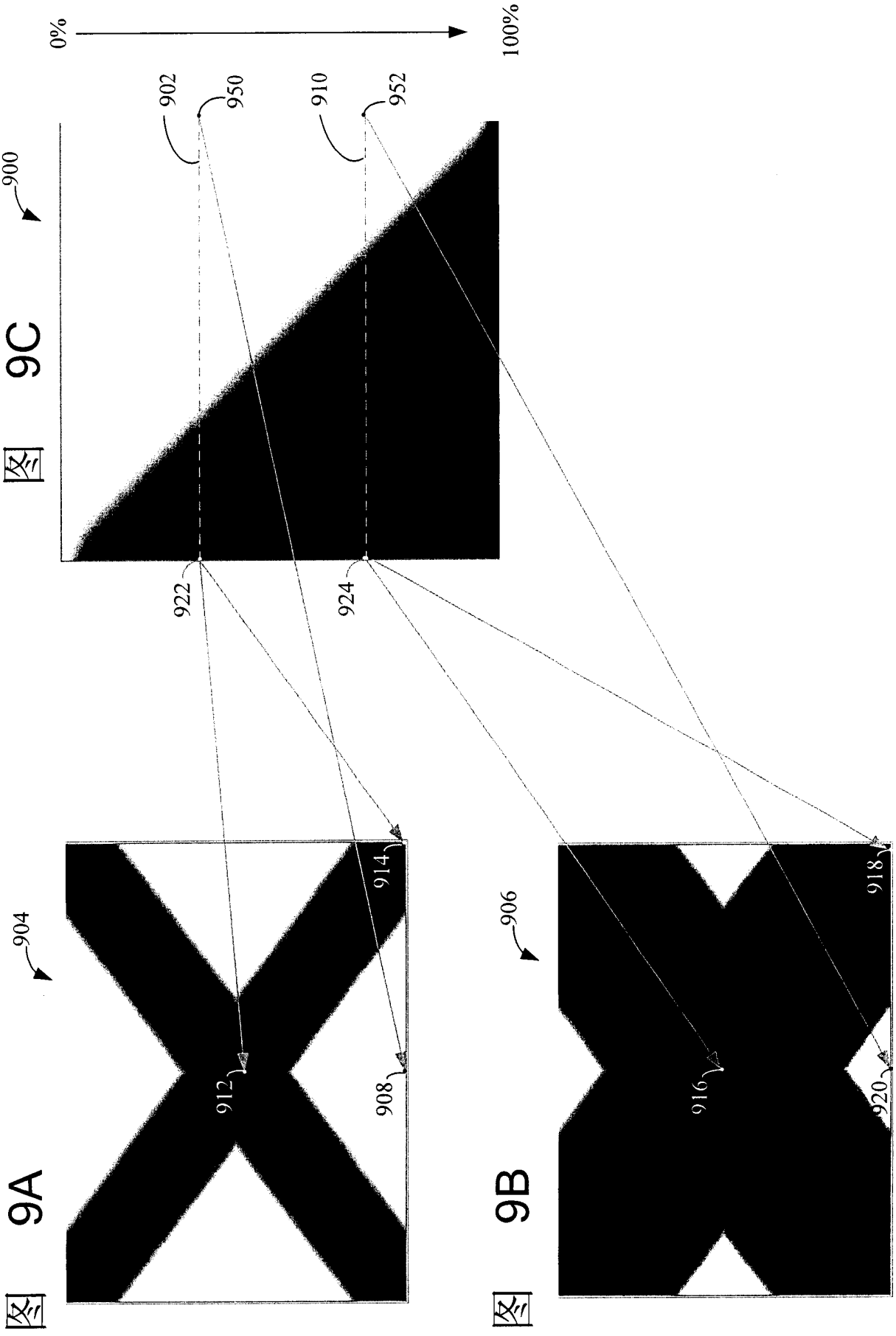


图 9C

图 9A

图 9B

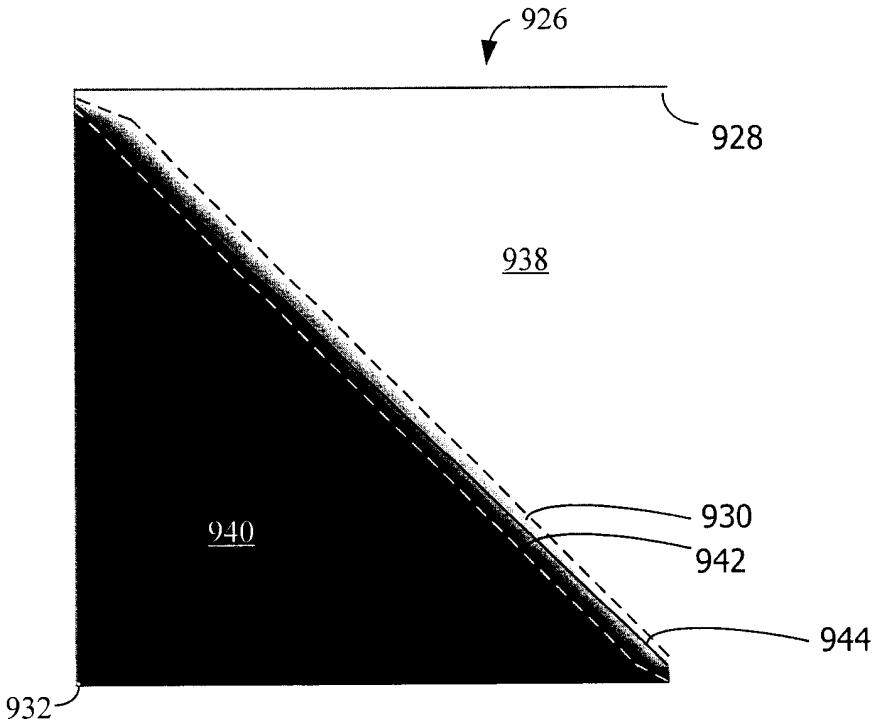


图 9D