第七章 视觉渲染

本章介绍了视觉渲染，它涵盖了视觉显示器通过虚拟世界生成器（VWG）界面显示的内容。第 3 章已经介绍了数学部分，我们可以表示出虚拟世界中的物体应该出现在屏幕上的位置。这部分基于几何模型，刚体变换和视点变换。我们接下来需要根据光传播，视觉生理学和视觉上的知识来确定这些物体应该如何出现。这些分别是第 4,5,6 章的主题。因此，视觉渲染是目前涵盖上述所有内容的顶层内容。

7.1 节和 7.2 节介绍了基本概念; 这是是计算机图形的核心，但也会出现特定的 VR 问题，主要针对综合形成的虚拟世界进行渲染的情况。 7.1 节解释了如何根据光源以及纯粹存在于虚拟世界中的材料的反射特性来确定应该出现在像素上的光。第 7.2 节解释了光栅化方法，它有效地解决了渲染问题，并广泛用于称为 GPU 的专用图形硬件。第 7.3 节解决了由光学系统缺陷引起的 VR 特定问题。第 7.4 节着重于延迟减少问题，这对 VR 来说至关重要，要保证虚拟物体在正确的时间出现在正确的位置。否则，许多副作用可能会出现，如 VR 不适，疲劳，对缺陷的适应性，或获得不具说服力的经历。最后，第 7.5 节解释了被捕捉的，而不是合成的虚拟世界的渲染。这包括从全景照片和视频形成的 VR 体验。

7.1 光线跟踪和着色模型

假设一个虚拟世界已经以三角形基元的形式被定义。此外，放置一双虚拟的眼睛以从一些特定的位置和方向对虚拟世界进行查看。使用第 3 章中的全变换链，每个三角形都被正确地放置在虚拟屏幕上（如图 3.13 所示）。接下来的步骤是确定哪些屏幕像素被变换后的三角形覆盖，然后根据虚拟世界的物理原理照亮它们。

还必须检查一个重要的条件：对于每个像素，三角形是否对眼睛可见，还是会被另一个三角形的一部分阻挡？这种经典的可视性计算问题极大地使渲染过程复杂化。一般问题是确定虚拟世界中的任何一对点，连接它们的线段是否与任何物体（三角形）相交。如果发生交叉，则两点之间的视线可视性被阻止。两种主要渲染方法的主要区别是如何处理可视性。

物体顺序 vs 图像顺序

对于渲染，我们需要考虑物体和像素的所有组合。这表明了一个嵌套循环。解决可视性的一种方法是迭代所有三角形的列表并尝试将每个三角形渲染到屏幕上。这被称为物体顺序渲染，并且是第 7.2 节的主题。对于落入屏幕视场中的每个三角形，仅当三角形的相应部分比目前为止渲染的任何三角形更接近眼睛时才更新像素。在这种情况下，外部循环遍历三角形，而内部循环遍历像素。另一种方法称为图像顺序渲染，它颠倒循环的顺序：遍历图像像素，并且对于每一个像素，确定哪个三角形会影响其 RGB 值。为了实现这一点，进入每个像素的光波路径通过虚拟环境被追踪。这种方法将首先被介绍，其中的许多组件也适用于物体顺序渲染。

光线追踪

为了计算像素处的 RGB 值，从放置在虚拟世界中的屏幕上的焦点通过像素的中心绘制观察光线; 如图 7.1。 该过程分为两个阶段：

1.*光线投射*，定义了观察射线并计算虚拟世界中所有三角形之间的最近交点。

2.*阴影*，根据照明条件和交点处的材料属性计算像素 RGB 值。

第一步完全基于虚拟世界几何。 第二步使用虚拟世界的模拟物理。 物体的材料属性和光照条件都是人工的，并且可以被选择来产生所需的效果，无论是现实还是幻想。请记住，

最终的判定者是通过感知过程解读图像的用户。

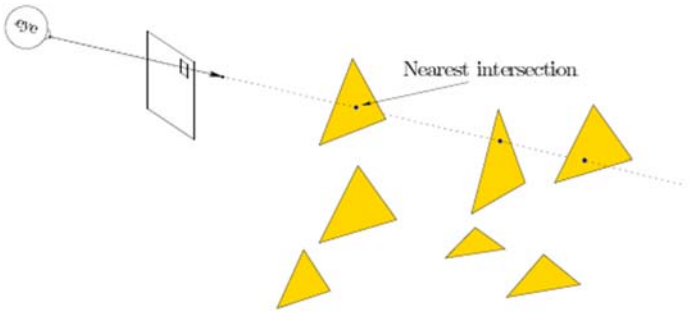

图 7.1：光线跟踪方法的第一步称为光线投射，它扩展了与图像上特定像素相对应的观察光线。 射线从焦点处开始，焦点是眼睛变换 T_{eye} 被应用后的原点。我们的任务是确定虚拟世界模型的哪一部分是可见的。它是观察射线和所有三角形集合之间最接近的交点。

光线投射

如果我们忽略计算性能，计算观察光线在离开图像像素后碰到的第一个三角形（图 7.1）很简单直接。从三角形坐标，焦点和射线方向（矢量）开始，封闭形式的解决方案包含来自解析几何的基本操作，包括点积，交叉积和平面方程[320]。对于每个三角形，必须确定射线是否与其相交。如果不是，则考虑下一个三角形。如果相交，那么仅当交叉点比迄今为止遇到的最近交叉点更近时，交叉点才被记录为候选解决方案。在考虑所有三角形之后，将最近的交点将被找到。虽然这很简单，但将三角形排列到 3D 数据结构中效率更高。这样的结构通常是分层的，因此许多三角形可以通过快速坐标测试来消除。流行的例子包括 BSP 树和 Bounding Volume Hierarchies [42,85]。对几何信息进行排序以获得更高效的算法通常被归类为计算几何[54]。除了从快速测试中消除许多三角形外，许多计算光线三角交点的方法也被开发以减少操作次数。其中最受欢迎的是 Möoller-Trumbore 相交算法[217]。

朗伯阴影

现在考虑点亮每个像素的情况并回顾 4.1 节中光的基本行为。虚拟世界模拟现实世界的物理学，其中包括光谱功率分布和光谱反射功能。假设一个点光源被放置在虚拟世界中。使用 6.3 节的三原色理论，其光谱功率分布可用 R，G 和 B 值充分表示。如果观察射线如图 7.2 所示撞击表面，那么物体应该如何出现？阴影模型考虑了关于光谱反射函数的假设。最简单的情况是朗伯阴影，其中观察射线撞击表面的角度与所得到的像素 R，G，B 值无关。这对应于漫反射的情况，适用于“粗糙”表面（回忆图 4.4）。表面相对于光源的角度是至关重要的。

设 n 是外表面法线，并且 ℓ 是从表面交点到光源的矢量。假设 n 和 ℓ 都是单位矢量，令 θ 表示它们之间的角度。点积 $n \cdot \ell = \cos\theta$ 表示由于表面相对于光源的倾斜而产生的衰减量（0 和 1 之间）。思考三角形的有效面积是如何由于倾斜而减小的。朗伯阴影模型下的像素按如下公式点亮

$$\begin{aligned} R &= d_R I_R \max(0, n \cdot \ell) \\ G &= d_G I_G \max(0, n \cdot \ell) \\ B &= d_B I_B \max(0, n \cdot \ell), \end{aligned} \tag{7.1}$$

其中（d_R，d_G，d_B）表示材料的光谱反射特性（三角形），（I_R，I_G，I_R）表示光源的光谱功率分布。 在典型的白光情况下，$I_R = I_G = I_B$。 对于白色或灰色材质，我们也有 $d_R = d_G = d_B$。

使用矢量符号，可以将（7.1）压缩成

$$L = dI \max(0, n \cdot \ell) \tag{7.2}$$

其中 L =（R，G，B），d =（d_R，d_G，d_B）和 I =（I_R，I_G，I_R）。 假定每个三角形都位于物体的表面上，而不是物体本身。 因此，如果光源在三角形后面，那么三角形不应该被照亮，因为它背对着光线（它不能从后面照亮）。 为了处理这种情况，最大函数出现在（7.2）中以避免 n • ℓ <0。

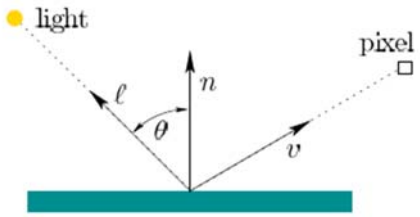

图 7.2：在朗伯阴影模型中，到达像素的光线取决于入射光与表面法线之间的角度 θ，但与视角无关。

Blinn-Phong 阴影

现在假设物体是“闪亮的”。 如果它是完美的镜像，那么只有当它们完全对齐时，来自光源的所有光才会反射到像素；否则，没有光线会反射。 如果 v 和 l 相对于 n 形成相同的角度，则会发生这种全反射。 如果两个角度接近但不完全匹配呢？ Blinn-Phong 阴影模型提出一定量的光线会被，这取决于表面闪光的量以及 v 与 l 之间的差异[24]。见图 7.3。等分线 b 是平均 l 和 v 后获得的矢量：

$$b = \frac{\ell + v}{\|\ell + v\|}. \tag{7.3}$$

使用压缩矢量标记，Blinn-Phong 着色模型将 RGB 像素值设置为

$$L = dI \max(0, n \cdot \ell) + sI \max(0, n \cdot b)^x. \tag{7.4}$$

由于漫反射和镜面反射两种成分，这种叠加考虑了阴影。 第一项只是朗伯阴影模型（7.2）。 随着 b 变得更接近于 n，第二部分导致越来越多的光被反射。 指数 x 是表示表面光泽度的材料属性。较低的值（例如 x = 100）表示轻微的光泽度，而 x = 10000 会使表面几乎像镜子一样。该着色模型不直接对应于光与表面之间相互作用的物理学。这只是一种方便而有效的启发式方法，但在计算机图形学中被广泛使用。

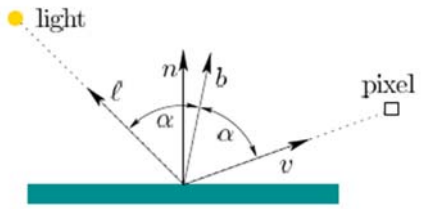

图 7.3：在 Blinn-Phong 阴影模型中，到达像素的光线取决于法线 n 与 l 和 v 的等分线 b 之间的角度，如果 n = b，则获得理想的反射，就像一面镜子。

环境阴影

另一个启发式想法是环境阴影，它会导致在不被光源照亮的情况下物体发光。这照亮了落入所有灯光阴影中的表面；否则，它们将完全变黑。在现实世界中，不会发生光线在物体

之间互相反射以照亮整个环境。到目前为止，这种传播在阴影模型中还没有被考虑到，因此需要黑客来修复它。添加环境阴影

$$L = dI\ \max(0, n \cdot \ell) + sI\ \max(0, n \cdot b)^x + L_a, \tag{7.5}$$

其中 L_a 是环境光分量。

多个光源

通常，虚拟世界包含多个光源。 在这种情况下，来自每个像素的光在像素处相加合并。对于 N 个光源的结果是

$$L = L_a + \sum_{i=1}^{N} dI_i\ \max(0, n \cdot \ell_i) + sI_i\ \max(0, n \cdot b_i)^x, \tag{7.6}$$

其中 I_i，l_i 和 b_i 对应于每个源。

BRDFs

迄今为止提出的着色模型由于其简单性和高效率而被广泛使用，尽管它们忽略了大多数物理学。 为了以更精确和一般的方式考虑阴影，建立了双向反射分布函数（BRDF）[231]; 见图 7.4。 θ i 和 θ r 参数分别表示相对于表面的光源角度和观察光线角度。 Φ i 和 Φ r 参数的范围是从 0 到 2 π，代表在向下看表面时光线和观察矢量所形成的角度（矢量 n 指向你的眼睛）。

BRDF 是如下形式的一个函数

$$f(\theta_i, \phi_i, \theta_r, \theta_i) = \frac{\text{radiance}}{\text{irradiance}}, \tag{7.7}$$

其中辐亮度是从方向 θ r 和 Φ r 从表面反射的光能，辐照度是从方向 θ i 和 Φ i 到达表面的光能。 这些被表示为不同级别，大致相当于一个无限小的表面补丁。通俗地说，它是出射光量与表面上一点入射光量的比值。 以前的着色模型可以用简单的 BRDF 表示。 对于朗伯阴影，BRDF 是恒定的，因为表面在所有方向上均等地反射。 BRDF 及其扩展可以考虑更为复杂和物理上正确的照明效果，并具有各种各样的表面纹理。 请参阅[5]的第 7 章以获得广泛的报道。

图 7.4：双向反射分布函数（BRDF），从所有可能的角度指定了入射和出射光能量的比率。

全局照明

回想一下，引入环境阴影项（7.5）是为了防止光源阴影中的表面显示为黑色。 解决这个问题的计算密集但适当的方法是计算光在虚拟世界中如何反射物体之间的物体。 通过这种方式，物体就会像现实世界中那样从被其他反射的光线间接照亮。不幸的是，这有效地将所有物体表面转变为潜在的光源。 这意味着光线追踪必须考虑多重反射。 这需要考虑从光源到视点的分段线性路径，其中每个弯曲对应于反射。 为通常需要考虑的反弹次数设定一个上限。 简单的朗伯和 Blinn-Phong 模型经常被使用，但更普遍的 BDRF 也很常见。 可以增加实际值的计算水平，但计算时间会相应增加。

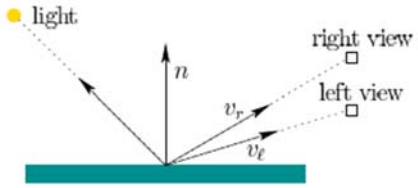

图 7.5：由于右眼和左眼的视点不同，光泽表面变得复杂。使用 Blinn-Phong 阴影模型，镜面反射对于每只眼睛应具有不同的亮度级别。为了与现实世界的行为保持一致，效果可能难以匹配。

VR 特有的问题

VR 继承了计算机图形学中的所有常见问题，但也包含独特的挑战。第 5 章和第 6 章提到了增加的分辨率和帧率要求。 这为降低渲染复杂性施加了很大的压力。此外，对于屏幕上的图形运行良好的许多启发式方法在 VR 中可能是明显错误的。高视角，分辨率，不同视角和立体图像的组合可能带来新问题。例如，图 7.5 展示了立体视图与立体视角的不同点如何影响光泽表面的外观。一般而言，一些渲染结果甚至可能导致 VR 不适。在本章的其余部分中，将会越来越多地讨论 VR 独有的复杂问题。

7.2 光栅化

光线投射操作很快成为瓶颈。对于 90Hz 的 1080p 图像，每秒需要执行 180 多万次，并且将针对每个三角形执行光线-三角形相交测试（尽管 BSP 等数据结构会快速消除许多考虑因素）。在大多数情况下，从这种图像顺序渲染切换到物体顺序渲染效率更高。本例中的物体是三角形，所产生的过程称为光栅化，这是现代图形处理单元（GPU）的主要功能。在这种情况下，通过遍历每个三角形并尝试对三角形落在图像上的像素着色来渲染图像。主要问题在于该方法必须解决确定哪个部分（如果有的话）最接近焦点（大致为虚拟眼睛的位置）的不可避免的问题。

解决这个问题的一种方法是按照深度顺序排列三角形，以便最近的三角形是最后一个。这使得三角形可以按照从后往前的顺序在屏幕上绘制。如果它们被正确地排序，那么任何后来渲染的三角形都会正确地将以前渲染的三角形的图像撞在相同的像素上。三角形可以一个一个绘制，完全不用考虑确定哪个最接近的问题。这被称为 Painter 的算法。但是，主要的缺陷在于深度循环的潜在存在，如图 7.6 所示，其中三个或更多三角形无法按照 Painter 算法的任何顺序正确渲染。一种可能的解决方法是检测这种情况并分割三角形。

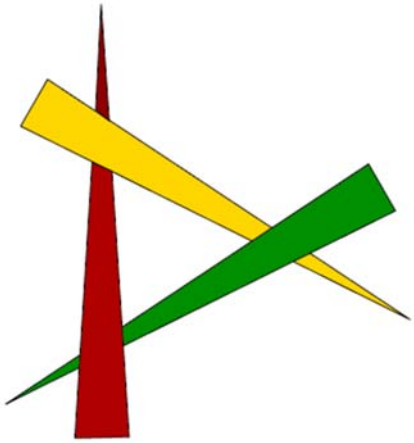

图 7.6：由于深度周期的可能性，对于距观察者的距离，物体不能按三维排序。每个物体都部分位于一个物体的前面，部分位于另一个的后面。

深度缓冲区

一种简单而有效的方法解决这个问题是通过维护一个深度缓冲区（也称为 z 缓冲区）来逐个像素地管理深度问题，对于每个像素，深度缓冲区记录三角形从焦点到在该像素处与三角形相交的射线的交点的距离。换句话说，如果这是射线投射方法，那就是沿着从焦点到交点的射线距离。使用这种方法，三角形可以以任意顺序渲染。该方法通常也适用于通过确定来自光源而不是视点的深度顺序来计算阴影的影响。靠近光线的物体会对其他物体产生阴影。

深度缓冲区在每个像素位置存储一个正实数（实际上是浮点数）。在渲染任何三角形之前，每个位置都会存储一个最大值（浮点无穷大），以反映每个像素上尚未遇到任何表面。渲染过程中的任何时候，深度缓冲区中的每个值都会记录最近渲染的三角形上的点与焦点的距离，以及图像中相应的像素。最初，所有深度都达到最大，以反映没有三角形被渲染。

通过计算完全包含它的图像的矩形部分来渲染每个三角形。这被称为边界框。通过变换三个三角形顶点来快速确定框，以确定 i 和 j（行和列索引）的最小值和最大值。然后对边界框内的所有像素执行迭代以确定哪些位于三角形内并因此应该被渲染。这可以通过形成如图 7.7 所示的三个边缘向量快速确定

$$\begin{aligned} e_1 &= p_2 - p_1 \\ e_2 &= p_3 - p_2 \\ e_3 &= p_1 - p_3. \end{aligned} \tag{7.8}$$

点 p 位于三角形的内部当且仅当

$$(p - p_1) \times e_1 < 0\ ,\ (p - p_2) \times e_2 < 0\ ,\ (p - p_3) \times e_3 < 0, \tag{7.9}$$

其中×表示标准向量叉积。 这三个条件确保 p 在每个边向量的“左边”。

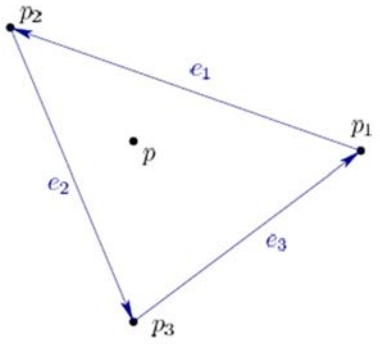

图 7.7：如果 p 在三角形内部，那么它必须在每个边向量 e_1，e_2 和 e_3 的右边。 重心坐标指定三角形中每个点 p 的位置作为其顶点 p_1，p_2 和 p_3 的加权平均值。

重心坐标

随着每个三角形的渲染，来自它的信息将从虚拟世界映射到屏幕上。这通常使用重心坐标来实现（见图 7.7），该坐标将三角形内部的每个点表示为三个顶点的加权平均值：

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 \tag{7.10}$$

其中 $0 \leqslant \alpha_1, \alpha_2, \alpha_3 \leqslant 1$ 且 $\alpha_1 + \alpha_2 + \alpha_3 = 1$。越接近于顶点 p_i，权重 α_i 越大。 如果 p 位于三角形的质心，则 $\alpha_1 = \alpha_2 = \alpha_3 = 1/3$。 如果 p 位于边上，则相反的顶点权重为零。 例如，如果 p 位于 p_1 和 p_2 之间的边上，那么 $\alpha_3 = 0$。如果 p 位于顶点 p_i，那么 $\alpha_i = 1$，另外两个重心坐标为零。

使用 Cramer 规则计算坐标以求解所得到的线性方程组。 特别地，对于 i 和 j 的所有组合，设 $d_{ij} = e_i \cdot e_j$。 此外，让

$$s = 1/(d_{11}d_{22} - d_{12}d_{12}). \tag{7.11}$$

坐标由下式给出

$$\begin{aligned} \alpha_1 &= s(d_{22}d_{31} - d_{12}d_{32}) \\ \alpha_2 &= s(d_{11}d_{32} - d_{12}d_{31}) \\ \alpha_3 &= 1 - \alpha_1 - \alpha_2. \end{aligned} \tag{7.12}$$

可以将相同的重心坐标应用于 R^3 中的模型上的点或图像平面上得到的 2D 投影点（具有 i 和 j 坐标）。换句话说，α_1，α_2 和 α_3 是指在第 3.5 节的整个变换链之前，期间和之后模型中的相同点。

此外，考虑到重心坐标，可以通过简单地确定 $\alpha_1 \geqslant 0$，$\alpha_2 \geqslant 0$ 和 $\alpha_3 \geqslant 0$ 来替代（7.9）中的测试条件。如果任何重心坐标小于零，则 p 必须位于三角形之外。

映射表面

重心坐标为在三角形上线性内插值提供了一种简单而有效的方法。最简单的情况是 RGB 值的传播。假设使用第 7.1 节的着色方法在三个三角形顶点计算 RGB 值。这产生对从 1 到 3 的每个 i 的（R_i，G_i，B_i）值。对于具有重心坐标（α_1，α_2，α_3）的三角形中的点 p，内部点的 RGB 值被计算为

$$\begin{aligned} R &= \alpha_1 R_1 + \alpha_2 R_2 + \alpha_3 R_3 \\ G &= \alpha_1 G_1 + \alpha_2 G_2 + \alpha_3 G_3 \\ B &= \alpha_1 B_1 + \alpha_2 B_2 + \alpha_3 B_3. \end{aligned} \tag{7.13}$$

该物体不需要在整个三角形上保持相同的属性。使用纹理贴图，可以在表面上传播重复的图案，例如瓷砖或条纹[41]。

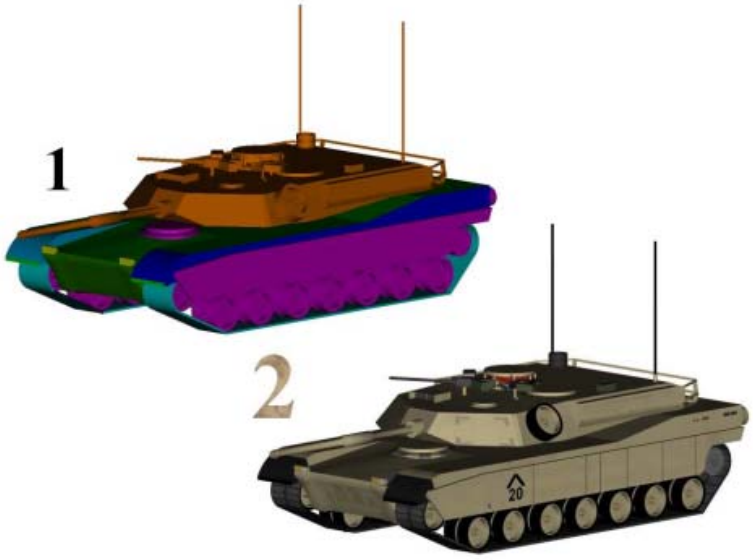

图 7.8 纹理贴图：将简单的图案或是整幅图像映射在三角形纹理中，然后在图像上进行渲染，相比于直接使用模型中的三角形纹理，这样可以提供更多细节。（图源于 Wikipedia）

通过*纹理贴图*，一些重复的图案，如瓦片或条纹等可以在物体表面进行传播[41]，如图 7.8 所示。更普遍的说，任何数字图像都可以映射到三角形上。重心坐标指的是图像中可以影响像素的点。图像，或是说“纹理”可以视为被绘制在三角形上；此外，为了更加真实，使物体有遮蔽效果，可以额外添加光照和反射属性。

另一种可行方法是*法线贴图*，通过在三角形上人工改变表面法线来改变遮蔽的过程，尽管它不能在几何上实现。在 7.1 节中的阴影模型中就使用了法线。允许其变化的话，可以在物体上添加一个仿真的曲率，法线贴图的一个重要实例叫做凹凸贴图，通过不规则地扰动法线使得平坦的表面变得粗糙。如果法线具有纹理的话，那么在计算阴影之后，整个表面看上去会显得很粗糙。

走样

由于数字图像的离散化，可能会出现一些缺陷。5.4 节中提到过走样问题，由于像素密度不足，导致一条直线看上去是阶梯状的。图 7.10（a）展示了在图（7.9）中小三角形内选中的像素。点 p 一般对应像素中心，如图 7.10（b）所示。可以看出，可能会出现该点在三角形内部而整个像素不在的情况。你可能会注意到，由于显示器中使用了亚像素镶嵌技术，图 7.10 可能不能表示真实情况。为了得到更准确的结果，走样分析也应该考虑到这一点。

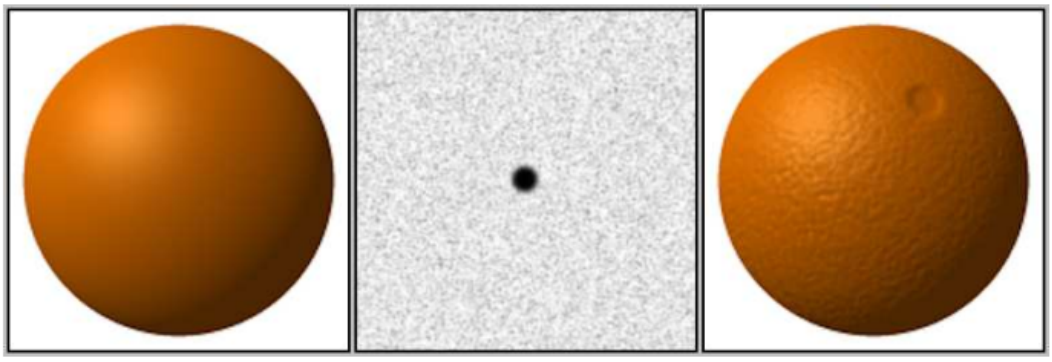

图 7.9：凹凸贴图：通过人工改变表面法线，阴影生成算法会产生一个看上去粗糙的表面（图源于 Brian Vibber）

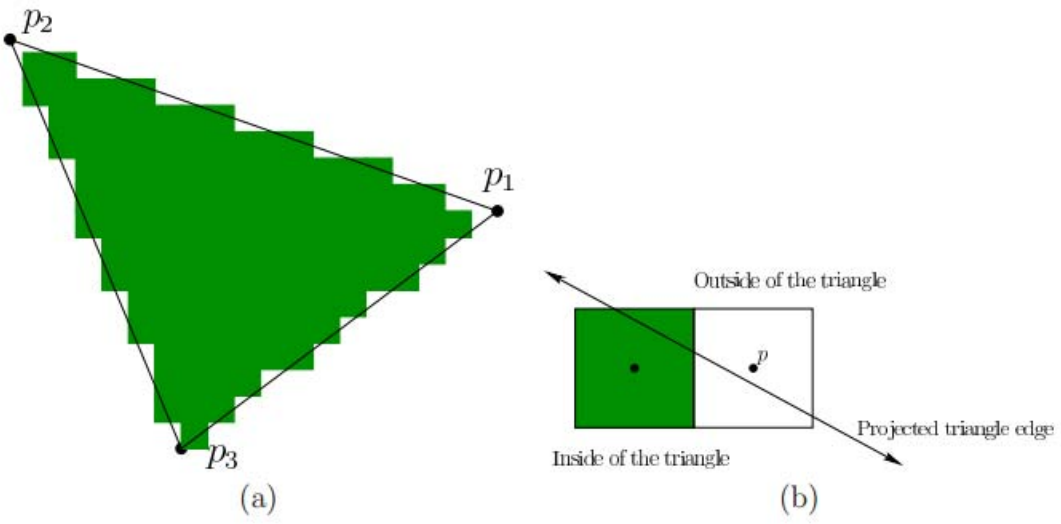

图 7.10：（a）光栅化导致走样现象：直线看上去像是阶梯状的 （b）根据像素的中心点 p 是否在三角形内部来选取像素

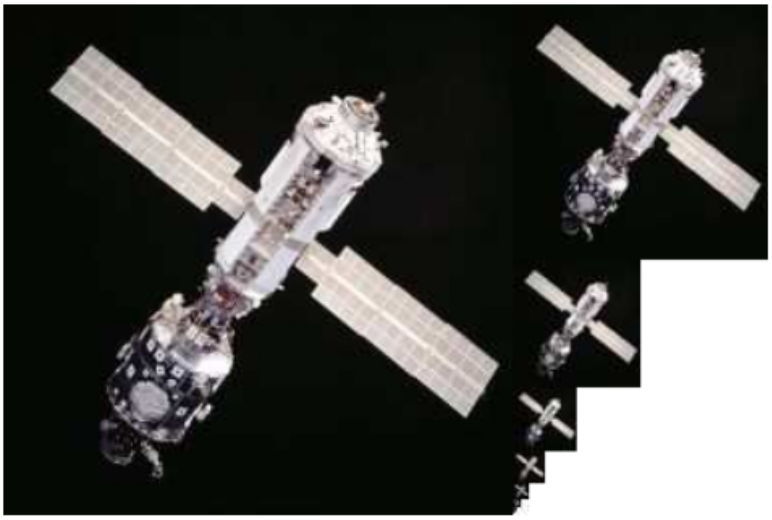

图 7.11：纹理贴图金字塔（mipmap），以多个分辨率存储纹理，使其可以适当缩放而不产生明显的走样现象。用于存储额外图像的空间通常仅为原始（最大）图像的 1/3（图源于 NASA，mipmap 由 Wikipedia 用户 Mulad 创建）

仅仅通过点 p 的坐标来决定其是否完全包含在三角形中的话，阶梯效应是无法避免的。更好的方法是根据三角形覆盖的像素区域来渲染像素。这样的话，它的值可以由像素区域内多个三角形混合而成。不幸的是，这需要*超采样*，意味着要以比像素密度高得多的密度来投射射线，从而估计三角形所覆盖的部分。这样的话代价会大大增加。通常，可以使用一种折中的方案，使用*多重采样抗锯齿*（或 *MSAA*）的方法，在较高密度下只计算其中的一些值。一般来讲，每个样本都要计算深度值，但阴影不用。

*空间走样*的问题是由纹理贴图引起的。将虚拟世界映射到屏幕上时，视变换会显著地减少原始纹理的大小和纵横比。如图 7.12 所示，用来表示纹理中的重复图案的分辨率可能会有所不足。在实践中，该问题通常通过预先计算和存储每个纹理的纹理贴图金字塔（mipmap）

来解决，如图 7.11 所示。通过对图像进行高密度采样及存储光栅化结果，纹理可以在多分辨率下计算。基于屏幕上三角形的大小和视点，选择适当的缩放纹理图像并贴图到三角形上，可以减少走样缺陷现象。

剔除

在实践中，许多三角形可以在渲染之前被快速地消除。这是预处理阶段一种叫做*剔除*的方法，可以显著提高性能，也可以提高帧率。剔除操作很大程度上取决于三角形的数据结构。如果所有三角形都按照层级结构排列的话，则可以通过单次坐标比较的方式消除上千个三角形。最基本的剔除方法叫做*视域体积剔除*，可以消除完全位于视锥体之外的所有三角形（回顾图 3.18）。对于 VR 头显，由于光学系统的限制，视锥体可能会有弯曲的横截面（如图 7.13）。在这种情况下，视锥体必须被替换为一个具有合适形状的区域。对于这种情况下的截锥体，简单的几何判断可以快速消除视域外的所有物体。例如，如果

$$\frac{\sqrt{x^2+y^2}}{-z} > \tan\theta \tag{7.14}$$

2θ指代视域的角度范围，点(x, y, z)在锥体的外面。或者，可以在 GPU 中使用*模板缓冲*来标记镜头视域外的所有像素。在每帧画面被渲染时，通过简单的判断可以快速消除这些问题。

另一种方式叫做*背面剔除*，用于消除具有远离焦点的外表面法线的三角形。当模型持续生成时，“背面”的部分没有被渲染的必要。此外，遮蔽剔除可用于消除模型中可能会被更近的物体遮挡住的部分。这可能有些复杂，因为它要再次考虑深度顺序问题。想要了解更多细节，可以参考[5]。

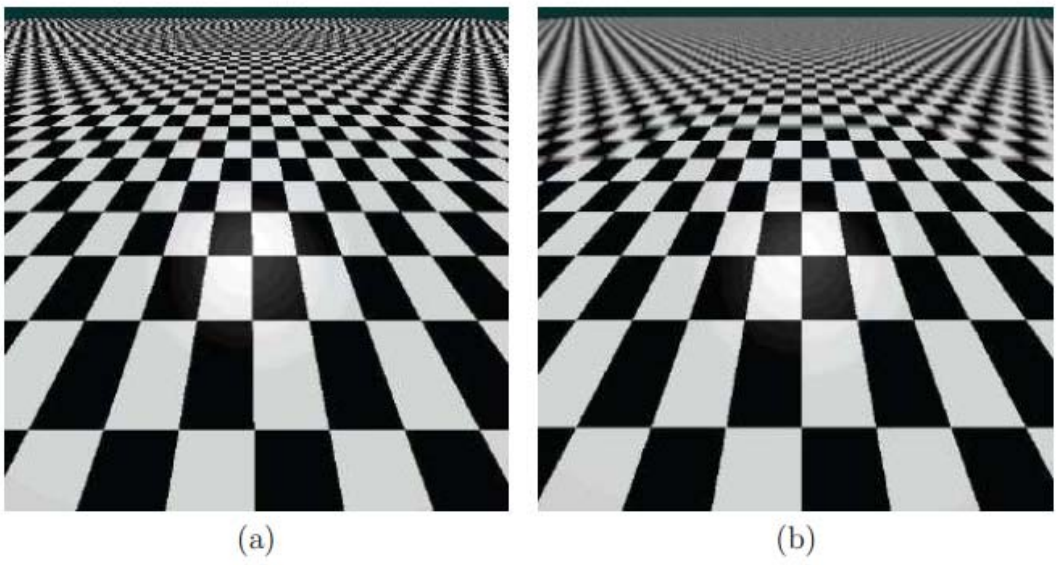

图 7.12：（a）由于透视变换，随着深度的增加，平铺的纹理会产生*空间走样*现象 （b）可以通过超采样来解决该问题

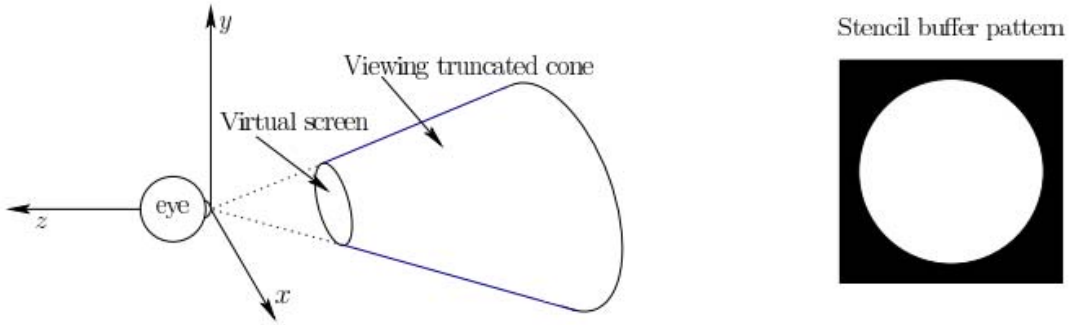

图 7.13：由于屏幕前方的光学系统，在圆形对称视图的情况下，视体将被截锥体代替。横截面形状与到每个眼睛视域的不对称性有关。（例如，鼻子会阻挡部分视域）

虚拟现实特有的光栅化问题

由于目前的分辨率远低于 5.4 节中所计算的视网膜显示的下限，所以由于走样造成的阶梯状问题应该会更严重。由于头部的运动导致视点不断改变，问题会变得更为严重。当用户试图注视边缘的时候，由于像素是否包含在三角形中是取决于视点的细微变化的，“阶梯”似乎更像是“自动扶梯”。作为正常感知过程的一部分，我们会被这种分散注意力的动作所吸引。对于立体的视点来说，甚至更糟：左右侧图像中的“自动扶梯”通常并不匹配。当大脑尝试将两幅图像融合成一幅连贯视图时，走样缺陷会生成强烈的运动不匹配的感觉。降低边缘的对比度并使用抗锯齿技术有助于缓解问题，但是对于 VR 来说，在显示器分辨率可以达到所需要的视网膜显示密度之前，阶梯问题一直会是一个很严重的问题。

VR 系统带来的深度感知的增强会引起一个更严重的问题。头部运动和立体视域都使用户感觉到的表面深度差异很小，这理应是一件好事，然而，在计算机图形学数十年的发展中都默认这样一种事实：将虚拟世界渲染到一个固定屏幕上时，如果从很远的距离观看，人们无法感觉到这些差异。虚拟现实会使纹理贴图看上去并不真实。例如，将地毯图片的纹理贴图到地板上时可能会让地板看上去更像是简单的涂漆。在现实世界中，我们自然可以将分辨地毯涂漆和真实的地毯。法线贴图也会出现相同的问题。由于在 VR 中两只眼睛的视线会汇聚到表面上，所以静态图像中看起来粗糙的表面经过凹凸贴图后在 VR 中有可能看上去完全平坦。因此，随着 VR 系统质量的提升，我们应该提高对渲染质量的要求，这意味着很多经典的方法应当被修改或舍弃。

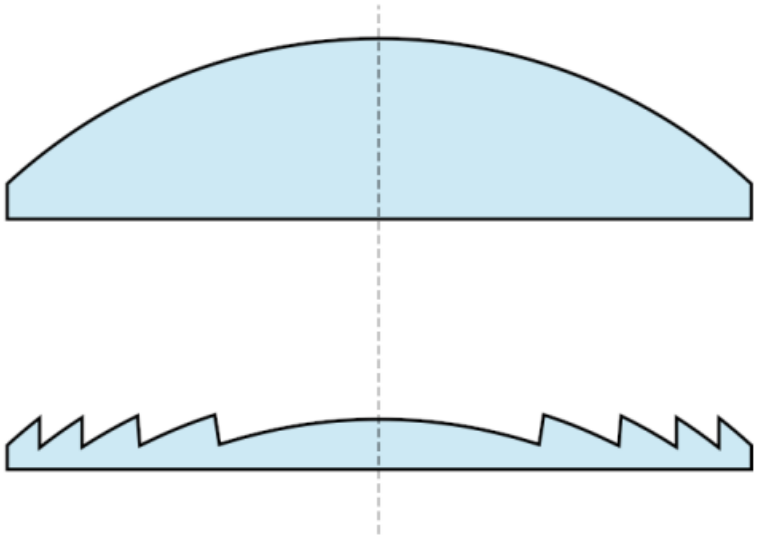

图 7.14：菲涅尔透镜，通过制造波纹表面来模拟简单的透镜，上方的凸面时由底部显示的菲涅尔透镜实现的

7.3 光学畸变的修正

回顾 4.3 节，对于高视场的光学系统，桶形畸变和枕形畸变是很常见的（如图 4.20）。通过 VR 头显的镜头观看时，通常会产生枕型畸变。如果图像不经过任何修正的话，那么虚拟世界看上去就会出现扭曲的现象。如果用户的头部来回转动的话，由于四周的变形比中心强烈，一些固定线条（如墙壁）的曲率会动态的改变。如果不加以修正，就没有一种静态物体的感觉，因为静态物体不应该会有动态变形。此外，这也有助于研究虚拟现实导致的一些疾病的成因，可能是由于在 VR 体验内感受到了四周异常的加速度。

那么，如何解决这个问题呢？现如今已经有很多相关的研究，可行的解决方案包括不同的光学系统及显示技术。例如，数字光处理（DLP）技术在不使用透镜的情况下直接将光投射到眼睛中。另一种实用的方法是使用菲涅尔透镜（如图 7.14），通过在较大的区域内用波纹或阶梯表面更精确地控制光线的弯曲；也可以设计成非球面的方案。例如，在 HTC Vive 的 VR 头显中就使用了菲涅尔透镜。但是它也存在副作用，当光沿着表面的突起散射的时候，会经常观察到眩光。

无论畸变的严重程度有多大，都可以通过软件进行修正。假设是默认畸变是循环对称的，这意味着畸变量仅取决于到镜头中心的距离，而不取决于中心的特定方向。即使镜头的畸变是标准的圆形对称，它也必须要放置在眼睛中央。一些头戴设备支持 IPD 调节，可以调节镜头之间的距离，使其可以匹配用户的眼睛。如果眼睛不在镜头中央，则会出现不对称畸变。这种情况并不能视为完美对称，因为随着眼睛的转动，瞳孔也会沿着球形弧面移动。随着镜头上瞳孔位置的横向变化，畸变会变得不对称。这促使厂家使用尽可能大的镜头来避免这种问题。另一个原因是，随着镜头和屏幕之间距离的变化，畸变也会变化。这种调整可以满足近视或远视用户的需求，正如三星 Gear VR 头显所做的。这种调整在双筒望远镜中也很常见，这就解释了为什么很多人在使用时不需要戴眼镜。为了正确地处理畸变问题，头戴设备应当能够准确地检测调整设置，并将这些因素考虑在内。

为了修正径向对称的畸变，我们假设将变换链 $T_{can}T_{eye}T_{rb}$ 应用到几何学中，生成视域体积范式，如 3.5 节所述。所有位于视锥体内的点的 x，y 坐标在-1 到 1 范围内取值。考虑用极坐标（r，θ）描述这些点：

$$\mathrm{r} = \sqrt{x^2 + y^2} \tag{7.15}$$
$$\theta = \mathrm{atan2(y, x)}$$

atan 表示 y/x 的反正切值。该函数通常通过编程计算，返回角度θ，范围在 0 到 2π之间（仅使用反正切是不可以的，因为还需要知道（x，y）所在的象限）。

现在，我们通过变化半径 r 来表示镜头的畸变，而由于对称性，方向θ并不会被影响。我们用 f 表示作用在正实数域上，变化半径为 r 的函数，用 r_u 表示未畸变的半径，r_d 表示畸变半径。枕形和桶形畸变通常可以使用奇数次幂的多项式来近似表示，我们定义 f 为：

$$r_d = f(r_u) = r_u + c_1 r_u^3 + c_2 r_u^5 \tag{7.16}$$

c1 和 c2 是常数。若 c1<0，则为桶形畸变；若 c1>0，则为枕形畸变。更高阶的项也可能会用到，例如在后面加上$c_3 r_u^7$项；然而，实际中并不需要考虑。

光学畸变的修正包含两个阶段：

1. 确定特定头显的径向畸变函数 f，将特定镜头置于屏幕前固定距离的位置。这是一个回归或曲线拟合问题，步骤包含测量许多点的畸变从而确定参数 c1,c2 等，然后取最佳的拟合结果。

2. 确定 f 的逆函数，使得可以在镜头产生畸变前将其应用到图像渲染上。f 的逆函数按理说应该可以抵消畸变带来的影响。

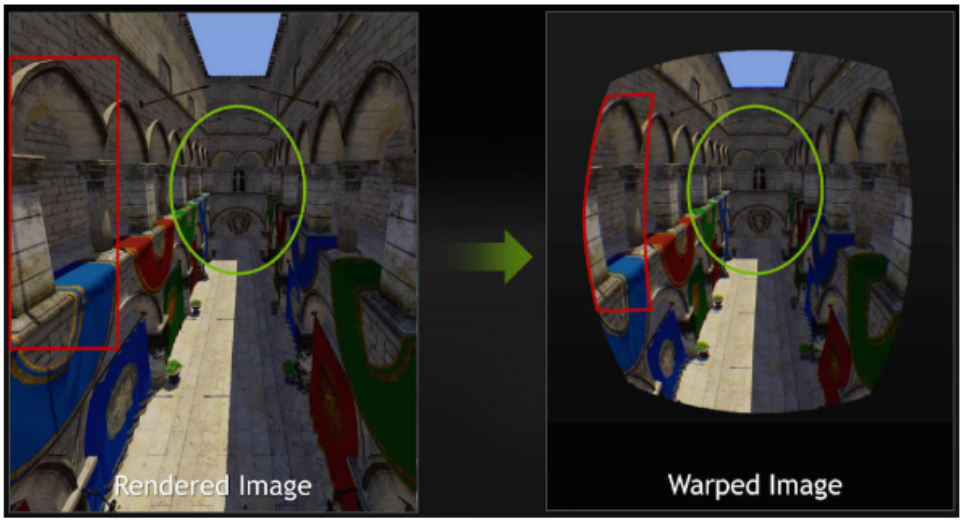

图 7.15：渲染图出现了桶形畸变。可以看到四周的分辨率有了明显的下降（图源于 Nvidia）

然而，多项式函数通常没有确定的或是闭合形式的逆函数，因此经常使用近似的结果。一种近似的方式如[118]所示：

$$f^{-1}(r_d) \approx \frac{c_1 r_d^2 + c_2 r_d^4 + c_1^2 r_d^4 + c_2^2 r_d^8 + 2c_1 c_2 r_d^6}{1 + 4c_1 r_d^2 + 6c_2 r_d^4} \tag{7.17}$$

或者，我们离线计算出精准的逆函数，然后存储在阵列中以便快速访问，每个头显只需要计算一次。使用线性插值有助于提高结果的精度。用牛顿法可以准确的计算出结果，初始值设置为交换坐标轴后的关于r_u的$f(r_u)$函数。

变换 f^{-1} 可以直接应用于透视变换，从而用非线性运算代替 T_p 和 T_{can}。通过现有的图形渲染管线，它可以作为一种后处理操作。图像变换的过程有时也被称为*畸变着色*，因为它可以作为 GPU 中的着色操作来实现；基于逐像素操作，7.2 节中计算出的光栅化图像可以经由式 7.17 或是 f^{-1} 的其他表示变换为变换后的图像。如果我们对枕形畸变进行补偿，那么所得到的图像可能会出现桶形畸变，如图 7.15.为了提高 VR 的性能，Nvidia GTX 1080 型 GPU 中使用了多分辨率着色。有一个问题是，由于变换后的图像中分辨率在边缘附近会明显的下降，导致浪费了原始图像中的阴影计算。相反地，我们可以在变换之前对图像进行渲染，将最终的变换结果考虑进去。对于会被变换所压缩的部分使用较低分辨率的图像进行渲染。

本节中介绍的方法也适用于径向对称的其他光学畸变。例如，可以通过通过分别变换红、绿、蓝色子像素来校正部分色差。每种颜色径向移动不同的量以补偿基于其波长发生的径向畸变。如果正在使用色差修正，那么如果将镜头从 VR 头显中取出，则可以明显的看出呈现给显示器的图像颜色没有完美对齐。渲染系统必须根据颜色创建像素位置的变形，使得在经过镜头后可以移动到正确的位置。

7.4 减小时延&提升帧率

VR 头显中的*系统延时*指的是从响应头部方向和位置运动到更新屏幕画面所需的时间。例如，假设用户正在注视虚拟世界中的某一固定特征点。此时如果头部偏向右侧，屏幕内对应的图像特征必须要立刻向左侧移动。否则，如果眼睛还在盯着该点，就会发现它有所移动。

这破坏了平衡的感觉。

一个简单的例子

我们通过下面的例子来感受延迟问题。设 d 为屏幕的像素密度，以 pixels/rad 为单位。令 ω 为头部转动的角速度，单位 rad/s；l 表示时延，单位 s。由于时延和角速度，头像偏移了 dωl 像素。例如，d=40pixels/rad，ω=50rad/s，l=0.02s，那么图像会产生 dωl=4pixels 的错误偏移。快速的头部旋转可能会有 300rad/s 的角速度，将会产生 24 像素的错误量。

完美的系统

我们做一个思想实验，考虑一个完美的 VR 系统。随着头部的移动，视点可以相应的改变视觉渲染。存在一个“神谕”，可以随时指示正确的头部位置和方向。VWG 可以一直维持虚拟世界中所有物体的位置和方向。视觉渲染系统维持着所有的视角和视域变换，并且根据着色模型，光栅化过程持续对屏幕的 RGB 值进行设定。进一步设想，屏幕本身是不断更新的，并不需要花费时间来切换像素。如图 5.4 节所示，屏幕具有视网膜级别的分辨率，并且为了匹配人类的感知，具有 7 个数量级的光线输出动态范围。在这种情况下，虚拟世界给予的视觉刺激应该可以与几何类似的物理世界的情景相对应。在时间和空间上不存在误差（尽管由于对照明、阴影、材料属性、色彩空间等的假设，可能并不能够与物理世界完全相同）

历史问题

在实践中，完美系统是不可能实现的。所有的这些操作都需要时间来传播信息和进行计算。在早期的 VR 系统中，*系统延时*经常会超过 100 毫秒。在 20 世纪 90 年代，延迟减少到了 60 毫秒。时延被认为是导致 VR 疾病的重要原因之一，因此这也是过去几十年 VR 广泛运用的一大障碍。一般来说，人类可以适应固定的延迟时间，这似乎对该问题有所缓解，但他们最后不得不重新调整到现实世界中去时，又会引发一系列问题。由于难以适应，时延如果是变化的话情况会更加糟糕[68]。幸运的是，由于最新一代的跟踪、GPU 和显示技术，时延不再是大多数 VR 系统的主要问题。现在的时延可以控制在 15 到 25 毫秒左右，甚至还可以通过跟踪系统的预测方法进行补偿，达到接近 0 的有效时延。因此，现如今其他因素对 VR 疾病和 VR 疲劳的影响更大，如体感和光学像差等。

减少时延的方法概述

以下的方法可以一起使用，从而减少时延并最小化剩余时延带来的副作用：

1. 降低虚拟世界的复杂度
2. 提高渲染管线的性能
3. 消除从渲染图像到切换像素之间的延迟
4. 预测估计未来的视点和状态
5. 移动或扭转渲染图以补偿上一时刻的视点错误和丢失的帧

我们会相继介绍上述的方法。

图 7.16：可以使用网格简化算法来减少模型的复杂度，同时保留重要的结构。此图显示的是由开源库 CGAL 制作的手模型的简化版（图源于 Fernando Cacciola）

简化虚拟世界

回顾 3.1 节，虚拟世界是由几何图元组成的，通常是三维网格中排列的三角形。对于每个三角形，变换和光栅化的过程是必须的，从而导致与三角形数量成正比的计算代价。因此，一个包含数千万三角形的模型要比一个仅由数千个三角形构成的模型要花费更长时间。在很多情况下，我们得到的模型比必要的要大得多，它们通常可以做的更小（用更少的三角形），而没有明显差异，这很像图像、音视频压缩的工作原理。为什么它们一开始会很大？如果模型是由现实世界的 3D 扫描中采集的，那么它包含了高度密集的数据。使用如 FARO Focus3D X 系列之类的采集系统可以采集外部的大型世界。其他的如 Matter 和 Form MFSV1，可以通过使物体在转盘上旋转的方法来进行采集。与相机一样，自动构建 3D 模型的系统专注于生成高度精确和密集的模型，从而尺寸也会最大化。即使使用纯合成的世界，Maya 或 Blender 等建模工具也会自动在曲面上构建高度精确的三角形网格。如果不关注对后面渲染过程造成的负担，模型可能会很快变得又笨又大。幸运的是，使用网格简化算法可以减少模型的大小，如图 7.16 所示。这样做的话，需要确保简化的模型在 VR 系统所有可能的视角中都有足够好的质量。在一些系统中，如 Unity 3D，减少模型中不同材质属性的数量也可以提高性能。

除了减少渲染时间之外，简化模型还会降低虚拟世界生成器（VWG）的计算需求。对于*静态世界*，VWG 在初始化后不需要执行任何更新操作。用户只是简单地观察这个固定的世界。对于*动态世界*，VWG 要维持虚拟世界的仿真，移动世界中的几何体以满足现实世界的物理规律。它需要处理角色的动作、坠落的物体、行驶的车辆、摇曳的树木等等。当物体之间相互接触产生反应需要用到碰撞检测方法。系统可以集成用于模拟运动定律的微分方程，以便物体能够随着时间正确的移动。这些问题将在第 8 章做介绍，现在只需理解 VWG 在每次进行渲染请求时都要保持虚拟世界的一致性。因此，VWG 具有和显示器或视觉渲染系统相同的帧率。每个 VWG 框架对应着常见时刻所有几何体的位置。VWG 每秒可以更新多少次？VWG 高帧率可以保持恒定吗？当 VWG 对世界进行更新的过程中提出渲染请求会发生什么？如果渲染模块不等待 VWG 完成就执行的话，一些物体就可能会被放置在错误的

位置，因为更新过程中只有部分物体随着更新。因此，理想情况下，系统应该等待，直到 VWG 帧更新结束再开始渲染。这要求 VWG 的更新速率至少应该与渲染过程一样快，并且要做好同步，以便于为渲染提供完整的、准确的 VWG 框架。

改善渲染性能

任何可以提高计算机图形学领域渲染性能的技术都适用于此。然而，我们必须避免在计算机显示器上察觉不到的副作用，这在 VR 中体现的更为明显。在 7.2 节中已经提到，VR 中的纹理和法线映射方法效率不高，未来几年可能会出现更多的差异。关于 VR 独有的改进，已在第 7.2 和 7.3 节中提到，模板缓冲区和多分辨率着色可通过利用由 VR 头戴显示器中的镜头引起的形状和畸变来改善渲染性能。进一步的改进可以在 GPU 中并行执行左右眼的光栅化，每个处理器使用一个处理器。这两个过程是完全独立的。这是专门设计针对 VR 的 GPU 的重要的第一步。

从渲染图像到切换像素

VWG 帧不连贯的问题也出现在渲染到显示的过程中：当需要将渲染图像扫描到显示器时，VWG 帧可能还没有完成。回想一下 5.4 节，大多数显示器都有一个滚动扫描，将位于显存中的光栅化图像的行逐一绘制到屏幕上。这是由电子束在模拟电视屏幕上点燃荧光粉的运动激发的。这个动作是从左到右，从上到下，就像我们用铅笔和纸写出一页英文文本一样。由于弯曲光束的磁性线圈中存在感应惯性，因此有几毫秒的时间称为 VBLANK（垂直消隐），其中光束从屏幕的右下方移动到左上方以开始下一帧。在此期间，光束将被关闭以避免在框架上画出对角线，因此，名称“消隐”。短消隐间隔也出现在每条水平线上，以使光束从右向左返回。

图 7.17：如果在发生显示扫描时将新的帧写入显存，则会出现裂痕，其中两个或更多帧的部分同时可见。

在数字显示时代，扫描过程是不必要的，但它仍然会存在并造成一些麻烦。假设显示器以 100 FPS 运行。在这种情况下，每 10ms 发出一次绘制新渲染图像的请求。假设 VBLANK 消耗 2ms，剩下的 8ms 花费在显示器上绘制线条。如果在 2ms 的 VBLANK 期间新的光栅化图像被写入显存，那么它将在剩余的 8ms 内正确绘制。也有可能通过激光束获得额外的时间[25,212]。但是，如果正在写入新图像并通过光束扫描出来的位置，则会发生裂痕，因为它看起来好像是屏幕被撕成碎片;见图 7.17。如果 VWG 和渲染系统以 300 FPS 生成帧，则 3 或 4 张图像中的一部分可能会出现在显示屏上，因为在扫描线条时图像会发生多次变化。解决此问题的一个解决方案是使用 VSYNC（发音为“vee sink”），这是一种防止显存写入 VBLANK 间隔之外的标志。

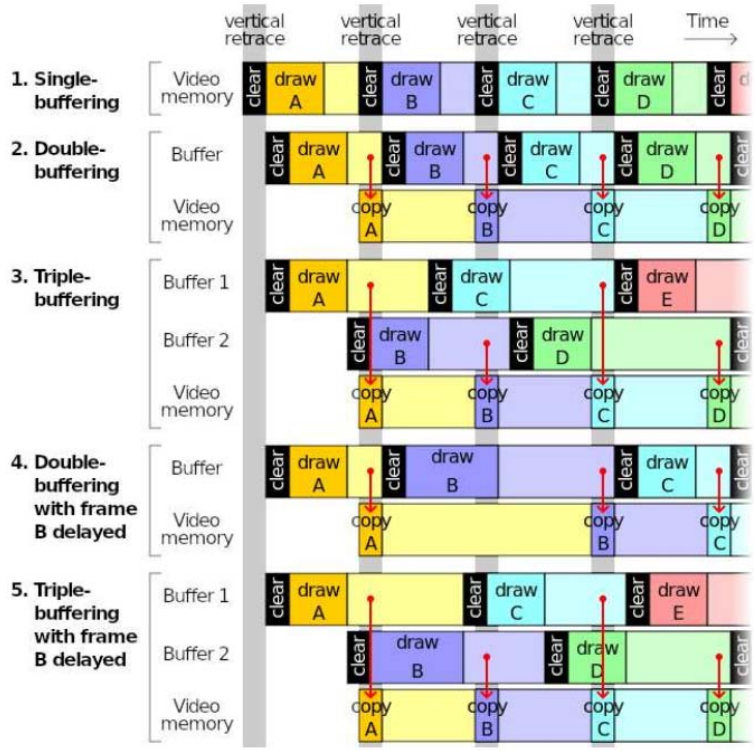

图 7.18：缓冲通常用于可视化渲染流水线以避免裂痕和丢失帧；然而，它引入了更多的延迟，这对 VR 是不利的。（图源于维基百科用户 Cmglee）

另一种避免裂痕的策略是缓冲，如图 7.18 所示。这种方法对于程序员来说很简单，因为它允许将帧写入未被扫描以输出到显示器的存储器中。它的一个副作用是增加了延迟。对于*双缓冲*，首先将一个新帧拖入缓冲区，然后在 VBLANK 期间传输到显存。通常很难控制产生帧的速率，因为操作系统可能暂时中断过程或改变其优先级。在这种情况下，三重缓冲是一种改进，允许更多时间渲染每一帧。为了避免裂痕和提供流畅的视频游戏性能，缓冲功能非常有用;然而这样导致延迟增加，这对 VR 来说是不利的。

理想情况下，显示器应该具有全局扫描，其中所有像素都在同一时间切换。这允许更长的时间间隔写入显存并避免裂痕。它还可以减少将第一个像素扫描到最后一个像素所需的时间。在我们的例子中，这是一个 8ms 的间隔。最后，显示器应尽可能减少像素切换时间。在智能手机的液晶显示屏上，切换像素可能需要长达 20ms 的时间;然而，OLED 像素可以在 0.1ms 以内切换。

预测研究

对于本节的其余部分，我们将考虑如何处理各种情况下的延迟。作为另一个实验，设想存在一个能够准确预测未来算命先生。有了这样的设备，应该可以消除所有延迟问题。我们想要问算命先生如下：

1.像素将在什么时候切换？

2.什么是所有虚拟世界模型的位置和方向？

3.哪个时候用户在哪里看？

让 t_s 来回答第一个问题。我们需要让 VWG 产生一个时间帧 t_s，然后在时间 t_s 对用户的视点进行视觉渲染。当像素在时间 t_s 切换时，刺激将在准确的时间和地点呈现给用户。在这种情况下，有效等待时间为零。

现在考虑实践中会发生什么。首先要注意的是，使用来自上述所有三个问题的信息意味

着在整个 VR 系统中显着的时间同步：所有操作都必须能够访问公共时钟。对于上面的第一个问题，如果计算机足够强大并且 VR 系统具有足够的来自操作系统的控制以确保 VWG 帧将以帧速率一致地产生和渲染，那么确定 t_s 应该是可行的。第二个问题对于静态虚拟世界来说很简单。在动态世界的情况下，对于根据可预测的物理规律移动的所有机构而言，可能很简单。然而，很难预测人类在虚拟世界中会做什么。这使第二个和第三个问题的答案变得复杂。幸运的是，延迟非常小，动量和惯性起着重要作用，详见第 8 章。匹配区域中的物体遵循真实世界的物理运动规律。这些动作根据第 9 章中介绍的方法进行检测和跟踪。虽然可能很难预测 5 秒钟内将要看到的位置，但可以非常准确地预测头部将在 20ms 内定位和定向的位置。你没有 20ms 的自由意志！相反，动量占主导地位，头部运动可以准确预测。有些身体部位，特别是手指，惯性较差，因此变得更难以预测;然而，这些并不像预测头部运动那么重要。视角仅取决于头部运动，延迟减少在这种情况下最为关键，以避免导致疲劳和 VR 晕眩的感知问题。

Perturbation	Image effect
$\Delta\alpha$ (yaw)	Horizontal shift
$\Delta\beta$ (pitch)	Vertical shift
$\Delta\gamma$ (roll)	Rotation about image center
Δx	Horizontal shift
Δy	Vertical shift
Δz	Contraction or expansion

图 7.19：基于视点变化的六种自由度渲染图像的扭曲。前三个对应于一个方向。剩下的三个对应一个位置变化。这些操作可以通过打开数码相机并观察图像在每个扰动下如何变化来可视化。

后渲染图像扭曲

由于预测过程中存在等待时间和不完善之处，因此在将帧扫描到显示器之前可能需要进行最后时刻的调整。这被称为后渲染图像扭曲[201]（它在最近的 VR 行业中也被重新发现并被称为时间扭曲和异步再投影）。在这个阶段，没有时间进行复杂的阴影操作;因此，只是对图像进行简单的变换。

假设图像已经被光栅化了一个特定的视点，由位置（x，y，z）和由偏航，俯仰和滚动（α，β，γ）给出的方位表示。如果对附近的观点进行光栅化处理，图像会有什么不同？基于视点的自由度，有六种类型的调整;见图 7.19。其中每一个都有一个未在图中指定的方向。例如，如果 Δα 是正的，这对应于视点的小的逆时针偏转，则图像水平地向右移动。

图 7.20 显示了一些图像扭曲的例子。大多数情况下需要渲染的图像比目标显示更大;否则，将不会有数据转移到扭曲的图像中;见图 7.20（d）。如果这种情况发生，那么最好从渲染的图像边缘重复像素，而不是将它们变成黑色[201]。

扭曲图像中的缺陷

由于方向变化导致的图像变形会产生正确的图像，因为它应该完全是从头开始为该方向渲染的图像（不考虑别名问题）。但是，位置的变化是不正确的。x 和 y 中的扰动不考虑运动视差（从 6.1 节回忆），但是需要知道物体的深度。 z 中的变化会产生类似不正确的图像，因为附近的物体应该比其他物体扩展或收缩更多。更糟糕的是，视点位置的变化可能导致可视性事件，其中物体的一部分可能仅在新视点中变为可见;见图 7.21。诸如方向图[250]和可视性复合体[252]之类的数据结构被设计用于维护这些事件，但通常不包括在渲染过程中。随着等待时间变短并且预测变得更精确，扰动量减少。精密的感知研究是必要的，它可以用来评估图像扭曲错误以及可察觉到的导致不适的情况。图像变形的一种替代方法是使用并行处理来对未来的几个视点进行采样并为所有视图渲染图像。然后可以选择最正确的图像，以大

大减少图像扭曲缺陷。

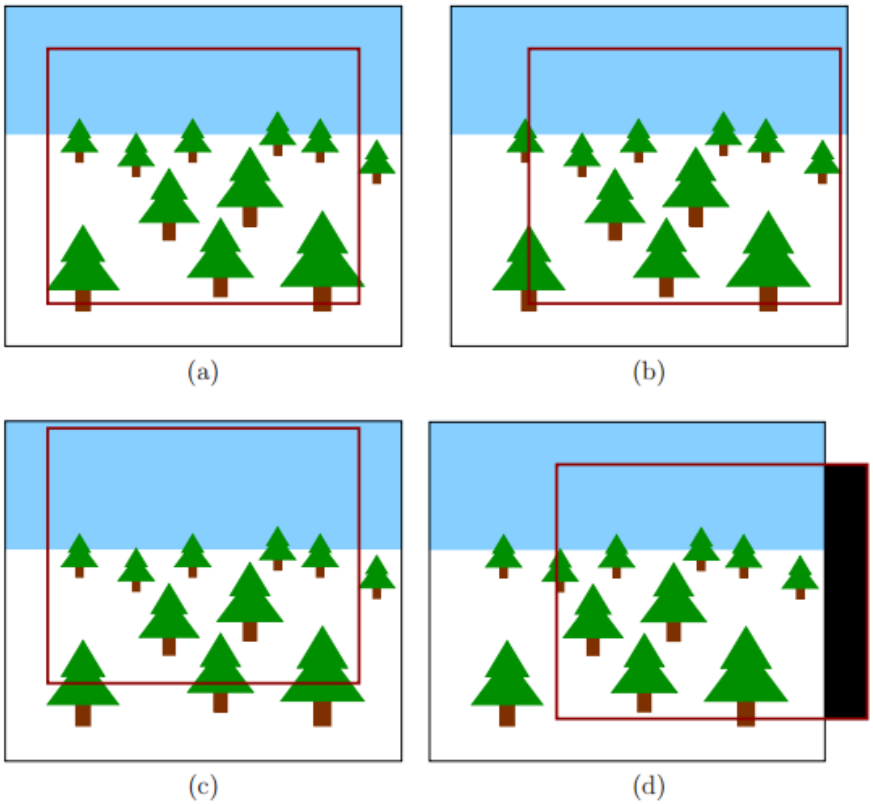

图 7.20：后渲染图像扭曲的几个示例：（a）在变形之前，更大的图像被光栅化。红色框根据光栅化时使用的视点显示要发送到显示器的部分；（b）负向偏航（将头部向右转）导致红色方框向右移动。图像似乎向左移动；（c）正节距（向上看）会使盒子向上移动；（d）在这种情况下，偏航过大，并且没有光栅化数据用于部分图像（该区域显示为黑色矩形）。

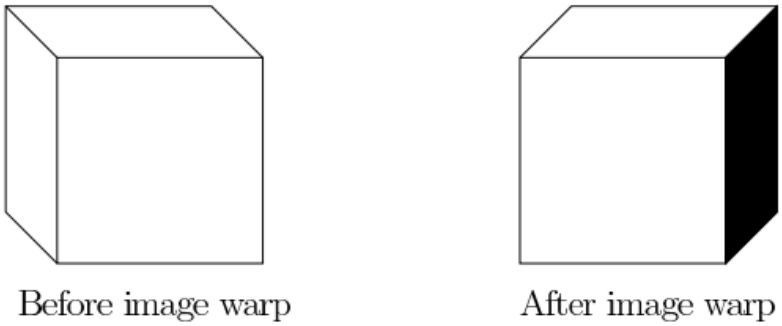

图 7.21：如果查看位置改变，则可能会遇到可视性事件。这意味着物体的一部分可能突然从新的角度变得可见。在这个示例中，视点中的水平移位揭示了最初隐藏的立方体的一侧。此外，立方体的顶部会改变其形状。

提高帧率

渲染后图像扭曲也可用于人为地增加帧速率。例如，假设弱电脑或 GPU 每 100 毫秒只产生一个光栅化图像。这会导致 10 FPS 时性能不佳。假设我们想将其提高到 100 FPS。在这种情况下，单个光栅化图像可以被扭曲以每 10ms 产生一帧，直到计算出下一个光栅化图像。在这种情况下，为每个正确渲染的光栅化图像插入 9 个变形帧。这个过程称为插入或补间，并且已经使用了一个多世纪（最早的例子之一是制作幻想曲，这在图 1.25（a）中有描述）。

7.5 沉浸式照片和视频

到目前为止，本章重点介绍了从几何模型综合构建的虚拟世界。数十年计算机图形学研究所开发的方法已针对这种情况进行了研究。不过，这种趋势最近发生了变化，人们更想捕捉真实世界的图像和视频，然后轻松嵌入到 VR 体验中。这一变化主要归功于智能手机行业，这导致成千上万的人们随身携带高分辨率相机。此外，3D 相机技术不断前进，除了颜色和光线强度之外，还提供距离信息。所有这些技术正迅速地融合到全景图的情况中，全景图包含从所有可能的观察方向获取的图像数据。当前的挑战是也捕获所有可能的观看位置和方向的区域内的数据。

(a) (b)

图 7.22：（a）截至 2015 年，Netflix 将在线电影流式传输到大型虚拟电视屏幕上，而用户似乎坐在起居室内。 （b）电影以帧为单位纹理映射到电视屏幕上。此外，注视指针允许用户查看特定方向以选择内容。

纹理映射到虚拟屏幕上

将照片或视频放入虚拟世界是纹理映射的扩展。图 7.22 显示了 Netflix 通过 Samsung Gear VR 头戴显示器提供在线电影串流的商业用途。虚拟屏幕是单个矩形，可以将其视为由两个三角形组成的简单网格。照片可以映射到虚拟世界中的任何三角形网格。在电影的情况下，每个帧被视为纹理映射到网格的照片。电影的帧率通常比 VR 头盔的低很多（回忆图 6.17）。例如，假设电影以 24 FPS 录制，头盔以 96 FPS 录制。在这种情况下，每个电影帧都会在头戴显示器上呈现四帧。大多数情况下，帧速率并不是完全可分的，这导致重复帧的数量在模式中交替。一个之前的例子被称为 3：2 下拉，其中 24 FPS 电影以 30 FPS 变换为 NTSC 电视格式。有趣的是，3D 电影（立体）体验甚至可以被模拟。对于头盔显示屏上的左眼，将左眼电影帧渲染到虚拟屏幕。类似地，右眼电影帧被呈现给头戴式显示器的右眼部分。其结果是用户将其视为 3D 电影，而不戴特殊眼镜！当然，她会戴着 VR 头盔。

捕捉更广阔的视野

映射到矩形上使得将用普通相机拍摄的照片或电影放入 VR 中更为简单。然而，VR 媒体本身允许扩展许多的体验。与真实世界中的生活不同，虚拟屏幕的大小可以扩展，而不会有任何显着的成本。为了填充用户的视野，将虚拟屏幕弯曲并将用户置于中心是有意义的。这种弯曲在现实世界中已经存在;例如 20 世纪 50 年代的 Cinerama 经验，如图 1.28（d）所示，以及现代曲面显示器。在极限情况下，我们会获得全景照片，有时称为光球。每秒显示许多照片，导致一部全景电影，我们可以称之为电影。

回顾第 4.5 节中相机的工作方式，无法在一个瞬间从单台相机捕捉光圈，这样的话，存

在两个明显的选择：

1.每次使用一台摄像机拍摄多张图像，每次指向不同方向，直至覆盖所有观看方向的整个球体。

2.使用多个摄像机，指向不同的观看方向，以便通过拍摄同步的图像覆盖所有方向。

第一种情况导致在计算机视觉和计算摄影中的图像拼接技术得到充分研究。通过将来自各种相机和时间的任意图像集拼接在一起，可以制作出硬版本。例如，从网上照片集合建立一个受欢迎的旅游网站光球。更常见的是，智能手机用户可以通过将朝向相机的方向指向足够的方向来拍摄光球。在这种情况下，软件应用程序动态构建光球，同时快速连续拍摄图像。对于硬版本，出现了一个困难的优化问题，其中需要在多个图像的重叠部分中识别和匹配特征，同时考虑未知的固有摄像机参数。必须考虑视角差异，光学像差，照明条件，曝光时间以及场景在不同时间的变化。在使用智能手机应用的情况下，正在使用相同的相机并且图像之间的相对时间很短，因此，任务要容易得多。此外，通过连续拍摄快速图像并使用内置的智能手机传感器，匹配重叠的图像部分要容易得多。这种手工制作的照相球中的大多数缺陷是由于用户无意中改变了照相机的位置，同时将其指向各个方向。

对于第二种情况，可以设计一系列相同的相机，以便覆盖所有观看方向;见图 7.23（a）。一旦设备被校准，相机的相对位置和方向就能被精确知道，将图像拼接在一起变得简单。尽管如此，由于照明或校准的变化，修正仍然需要，否则，缝合中的接缝可能可以察觉得到。这种权衡取决于相机的数量。通过使用许多相机，可以以相对较小的光学畸变进行非常高分辨率的拍摄，因为每个相机都会为光圈提供一个窄视场图像。另一方面，只有两台摄像机就足够了，例如理光 Theta S（图 7.23（b））。相机指向 180 度，鱼眼镜头能够捕捉大于 180 度的视角。这种设计大大降低了成本，但需要对两个捕获的图像进行重要的变形修正。

图 7.23：（a）360Heros Pro10 HD 是一台可以在相反方向安装 10 台 GoPro 摄像机以捕获全景图像的装备。（b）理光 Theta S 仅使用两台摄像机拍摄全景照片和视频，每台摄像机均提供一个视场大于 180 度的镜头。

映射到球体上

制图中众所周知的地图投影问题是将光球映射到屏幕上，然而，在 VR 中渲染光球时不会出现这种情况，因为它直接映射到虚拟世界中的球体上。所有可能的观察方向的球体映射到虚拟世界球体而没有畸变。为了直接使用纹理映射技术，虚拟世界球可以用统一的三角形近似，如图 7.24（a）所示。光球本身的存储方式不会降低某些地方的分辨率。我们不能简单地使用纬度和经度坐标来索引像素，因为极点和赤道之间的分辨率差异太大。我们可以使用与四元数使用指数（a，b，c）覆盖球体并且要求 a2 + b2 + c2 = 1 的方式类似的坐标。然而，相邻像素的结构（上，下，左，右）并不清楚。一个简单而有效的折中办法是将光球表示为六个方形图像，每个图像对应一个立方体的面。这就像一个六面 CAVE 投影系统的虚拟

版本。然后，每个图像可以很容易地映射到网格上，分辨率损失很小，如图 7.24（b）所示。

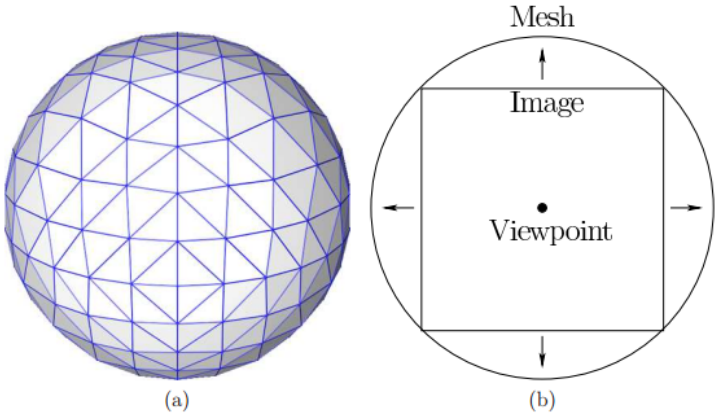

图 7.24：（a）光斑纹理映射到球体的内部，被建模为三角形网格。 （b）以六个图像的立方体形式存储的光球可以快速地映射到球体，分辨率损失相对较小；这里显示了横截面。

一旦光球（或电影球体）被渲染到虚拟球体上，渲染过程就非常类似于渲染后图像扭曲。呈现给用户的图像被变换为图 7.19 中描述的旋转情况。实际上，整个光栅化过程只能对整个球体执行一次，而渲染到显示器的图像则根据观察方向进行调整。进一步优化可以通过绕过网格并直接从捕获的图像形成光栅化图像来完成。

感知问题

观看光球或电影球时虚拟世界是否显示为“3D”？回想 6.1 节，还有更多的单眼深度线索，它们比立体线索更多。由于现代虚拟现实头戴式设备和单目深度线索的高视野，获得令人惊讶的沉浸式体验。因此，即使全景图像的相同部分呈现给双眼，它也可能比人们所期望的更“3D”。许多有趣的问题留待未来的研究关于全景的看法。如果左眼和右眼呈现不同的观点，那么虚拟球体的半径应该是多少，以获得舒适和逼真的观看效果？继续进一步，假设使用位置头部跟踪。这可能会提高观看舒适度，但由于视差不起作用，虚拟世界将显得更加平坦。例如，当头部左右移动时，更近的物体不会更快移动。可以对图像执行简单的变换，以便增强深度感知？有限的深度数据甚至可以从图像中自动提取，可以大大提高视差和深度感知？另一个问题是设计光球内的接口。假设我们想与球体内的其他用户分享经验，在这种情况下，我们如何看待插入球体的虚拟物体，如菜单或头像？虚拟激光指针如何工作以选择物体？

图 7.25：Figure Digital 的 Pantopticam 原型使用数十个相机来提高逼近更多视点的能力，从而可以模拟立体观看和位置变化引起的视差。

全景的光场

全景图像构造简单，但显然有缺陷，因为它们没有考虑从用户移动可能获得的任何视角环绕世界将如何出现。为了准确地确定这一点，理想的情况是捕获用户允许移动的任何观察体积内的整个光场能量。光场提供空间每个点处的光传播的光谱功率和方向。如果用户在穿着 VR 头盔时能够在物理世界中走动，那么这似乎是不可能完成的任务。一台摄像机如何在整个房间的同一时刻在所有可能的位置捕获光能？如果用户被限制在一个小区域内，那么光场可以被布置在球体上的一系列摄像机大致捕获，图 7.25 显示了这样一个原型。在这种情况下，可能需要数十个相机，并且使用图像变形技术来近似相机之间或来自球形内部的视点。为了进一步改善体验，光场照相机（也称为全光照相机）能够捕捉光线的强度以及它们在太空中行进的方向。这提供了许多优点，例如在光场已被捕获后将图像重新聚焦到不同的深度。

进一步阅读

VR 和计算机图形之间存在紧密的联系，因为两者都需要将可视信息推送到显示器上，然而，许多细微的差异出现了，而 VR 的发展也相当不够。对于基本的计算机图形学，许多文本提供了本章主题的额外报告，例如参见[202]。有关计算机图形的高性能，高分辨率渲染的更多细节，请参见[5]。除[5]外，BRDF 的全面报告见于[22]。

光线追踪范例可能需要重新设计 VR。从计算几何角度来看有用的算法背景可以在[336,42]中找到。有关光学畸变和校正背景，请参阅[46,117,130,199,326,330]。色差校正出现在[230]中。[33，299，317]中介绍了全景图的自动拼接。