第二章 全景视频呈现技术

2.1 HMD 的基本组成及典型设备

HMD 简介

图 2.1 各式各样的 HMD 设备

头戴式显示器 (Head Mount Display)，缩写为 HMD，指佩戴于头部的显示装置。HMD 有许多用途，包括游戏，航空，工程、医学等。

典型的 HMD 具有一个或两个小型显示器，并通过将透镜和半透镜嵌入眼镜（也称为数据眼镜）实现。显示单元是小型化的，可能包括阴极射线管（CRT）、液晶显示器（LCD）、硅上液晶（LCos）、有机发光二极管（OLED）等。一些供应商会采用多个微显示器来达到提高总分辨率和扩大视野的目的。

大多数 HMD 仅显示计算机生成的图像（CGI），也称为虚拟图像，或来自真实物理世界的实时图像。还有一部分 HMD 允许 CGI 叠加在真实世界的视图上，称为增强现实或混合现实（AR/MR）。真实世界视图与 CGI 的结合可以通过将 CGI 投射到部分反射镜并直接观察现实世界的方法来完成，这种方法通常称为光学透视；也可以通过接收来自摄像机的视频并以电子方式与 CGI 混合的形式完成。这种方法通常称为视频透视。

基本性能参数

a) 立体图像显示能力：为了让用户形成立体图像的错觉，双目 HMD 向每只眼睛输入不同的图像。在物体与眼睛距离较大时，物体在每只眼睛内形成的图像趋近相同；而对于距离较小的范围，两只眼睛的视角明显不同，所以通过 CGI 系统产生两个不同的视觉通道是有必要的。

b) 瞳距（IPD）：指两只眼睛之间的距离，瞳距的设定对于头戴式显示器而言非常重要。

c) 视场（FoV）：人类的 FoV 约为 220 度，但大多数的 HMD 远远低于此值。通常，更大的视场会带来更强的沉浸感和更好的情境感知。消费级别的 HMD 通常提供约 30-40° 的 FoV，而专业 HMD 提供的 FoV 在 60° 至 150° 之间。

d) 分辨率（Resolution）：在 HMD 中通常表示为像素总数或每度数的像素数（PPD）。60 PPD 通常被称为人眼上限分辨率。超过该分辨率，视力正常的人不会注意到增加的额外分辨率。目前的 HMD 通常只能提供 10-20 PPD，但微显示器的发展可以逐渐增加这一数值。

e）双目重叠，双眼共有的区域：双目重叠是视觉深度和立体感的基础，它允许人们感知物体的远近。人类的双目重叠约为 100°。而 HMD 提供的双目重叠程度越大，立体感越强。一般会以度数指定重叠范围或以百分比表明每只眼睛相对于另一只眼共同的虚拟 FoV 占比。

HMD 关键技术

想要营造真正令人沉浸的 VR 体验，需要攻克一系列技术难关：宽视角、高分辨率、像素低存留、高刷新率、全局显示、光学透镜和校准、低延迟，以及运动追踪。

视角越宽，沉浸感、临场感会越强，此外，尽管人们很难分辨出视线边缘的文字和图形，但这些视觉线索对移动、平衡、环境感知而言仍然至关重要。

随着视野变宽，像素将被拉伸，视野中每一度对应的像素将会减少，图像则变得粗糙。以 1K×1K 分辨率的显示屏为例，当它在 HMD 中，以 110 度视角展现给使用者时，像素密度只有普通电视机的七分之一，人眼上限分辨率的十分之一，这甚至还不如在 320×200 分辨率下的原始 PC 游戏清晰。若想得到较好的临场感，至少需要以 1080p 分辨率显示图像。

对于 HMD，由于双眼距离显示屏越近，它们之间的相对位移越快，因而如果像素存留时间较长，前庭动眼反射（VOR）将导致眼球移动时的图像变得模糊。如果不加以解决，在头部移动时，几乎无法清晰阅读文字，对 1K×1K 分辨率的 HMD 来说，像素存留时间不能超过 3 毫秒，而且像素密度越高，存留时间应越短，相对应的，刷新率就需要提高。在任何场景下都保持高刷新率并不容易。同时随着分辨率提高，需要有更强大的硬件和与之紧密配合的软件来支持高刷新率。

人眼对视觉上的异常非常敏感。有些 HMD 在静止时观看的图像尚可接受，但一摆头，临场感却荡然无存。HMD 需要控制焦距、畸变、像差等多种因素，这也是照相机为了获得高质量图片需要拥有复杂光学镜头的原因。而受空间和重量限制，HMD 中，每只眼睛前最多只能有一到两个镜片。对此，Oculus 重新发明了 HMD 的制造方式，利用计算机技术预先处理图像畸变，而不再倚赖造价高昂的光学镜片。

延迟是 HMD 技术的核心，高延迟是导致眩晕，破坏沉浸感的罪魁祸首。图像变化与用户实际移动之间的延迟接近 20 毫秒，就能产生相当不错的体验。对于 HMD，延迟存在于 6 个层面：a）运动捕获；b）数据和命令通过 USB 线缆传入 PC 或手机；c）游戏引擎响应命令后调动 GPU；d）GUP 生成一帧新图像；e）HMD 开始显示像素；f）图像完全显示。这一点在之后章节中仍会提到。

运动追踪技术同样是使用户相信自己真实地处于虚拟世界的关键技术。它可以通过追踪头部的运动状态实时更新渲染的场景，这与我们在真实世界中观看周围环境非常类似。高速的惯性测量单元（IMU）被用于快速的头动追踪。它结合了陀螺仪、加速度计（或磁力计，类似手机中使用的重力感应装置），可以精确测量转动的变化。头部运动追踪非常重要，因为人类的感知系统对运动非常敏感，如果在头部移动时图形显示出现延迟，那么就会破坏沉浸感，甚至引起恶心不适。因而，虚拟现实的 IMU 设备必须快速追踪头部运动，同时相应软件的性能也应匹配。只有当立体渲染和运动追踪结合得很好时，才能使得图形刷新帧率足够高，虚拟现实体验才能达到真正意义上的沉浸感。

就目前的情况来看，VR 行业的发展态势良好，包括 HypeVR、NextVR 在内的许多团队始终致力于 VR 各环节的技术研究。而像 Facebook、Samsung、HTC 等知名厂商也已着眼于 VR，并于近年带来了许多先进、便捷的产品。接下来便将对主流的 HMD 产品进行简要介绍。

主流 HMD —— 高端性能的主机+头盔

a）Vive-Pro

图 2.2 Vive-Pro 设备

- 单眼分辨率 1440 x 1600，双眼分辨率为 3K（2880 x 1600）
- 刷新率 90Hz
- 视场角 110 度
- 关键传感器：SteamVR 追踪、G-sensor 校正、gyroscope 陀螺仪、proximity 距离感测器、瞳距感测器

尽管相比初代产品，性能上提升了很多，但原先的HTC Vive并没有被抛弃，HTC为Vive和Vive Pro推出了无线适配器。正如人们所期望的那样，这个小插件可以夹在HMD后部，无需将接线连回PC端。此外，Vive Pro后部的新增配件和定位盘极大地提升了长期沉浸式体验的舒适感。

b） Pimax 8K VR

图 2.3 Pimax 8K

不出所料，从奥运广播测试平台引入8K等现象来看，8K也进入了VR世界。

虚拟现实领域对于8K的研究一直很少也很难进行，直至中国创业公司Pimax为其开辟了道路。该公司在2017年筹集了超过400万美元的资金，承诺向VR提供模块化方案，允许赞助商从一系列选项中选择硬件规格，包括无线，手柄和眼部追踪等相关的硬件。

实际上，Pimax并没有提供人们熟悉的电视屏幕（7,680 x 4,320分辨率）显示的8K，而是相对较低的7,680 x 2,160的分辨率。然而，这仍然是业界领先的分辨率方案（单目4K，90Hz刷新率）。但分辨率提高的同时，该公司也面临着8K VR内容、兼容的PC硬件紧缺的问题。

Pimax已经表明，对于那些追求更高分辨率VR的用户，Pimax 还研究了更强的 Pimax 8K X设备，这个版本经过特别设计，需要 NVIDIA GTX 1080 Ti来驱动，然而仅为使用该显卡，Pimax就需要为每台设备支付约1000英镑的成本。

除了8K分辨率的亮点，Pimax特隆风格的造型在同类产品中焕然一新。为了提升VR的沉浸效果，Pimax 8K采用了稍小于人类自然220度视场的200度FoV，是目前VR产品中最好的，同时减少了VR内容边缘的黑色边框。Pimax也承诺在正式出货之前实现15ms MTP的延迟。

c) Oculus Rift（DK2）

图2.4 Oculus Rift（DK2）设备

- 低延迟视觉跟踪系统
- 刷新率超过 75Hz
- 视场角 100 度
- 单眼分辨率 960×1080

d) HMD Odyseey

图2.5 HMD Odyseey设备

- 分辨率 2880×1600
- 视场角 110 度
- 前面板有两个摄像头，用于实现 inside-out 定位追踪以及现实空间数据的收集
- 配备的手柄在运动时配合头显上的摄像头进行数据采集，从而进行定位。其优点在于成本不高而且定位效果不错；缺点在于容易受环境因素影响。

主流 HMD —— 手机+头盔

a) Samsung New Gear VR

图2.6 Samsung Gear VR设备

- 视场角101度
- 关键传感器：加速度传感器，陀螺传感器，接近传感器

b) Google DayDream

图2.7 Google DayDream设备

- 视场角90度
- 配备无线控制器，内置陀螺仪，可检测方向、行动以及加速，实现位置追踪

主流 HMD —— 一体机

a) 联想 Mirage Solo

图 2.8　联想 Mirage Solo

VR 技术成为科技主流的一个明显标志就是更多的制造商将产品线扩大至虚拟现实领域。联想便是其中之一，其在 CES 上发布了与谷歌合作的 VR 一体机 Mirage Solo。

“无需 PC 或智能手机，没有杂乱的接线”，Mirage Solo 基于 Daydream 平台融合了运动跟踪技术，采用联想 Mirage 相机，可在 180 度 FoV 范围内拍摄 VR 视频。此外，Mirage Solo 搭载骁龙 835 芯片，配备分辨率为 2560 x 1440 LCD 液晶显示屏，续航时间达 7 小时。

更为重要的是，Mirage Solo 采用了 Google WorldSense 六自由度追踪定位系统，配备三自由度的控制器。WorldSense 不仅能追踪用户头部的旋转角度，而且还可以追踪整个身体在佩戴 Mirage Solo 时的移动情况。

与 Google 合作的另一创新点是 VR180，VR180 是以用户为中心的 180 度视频格式，观看时可以通过转头来切换视角，视频边界则以黑边显示，相当于 360 视频切掉了一半。这一理念是由大部分用户只观察 VR 视频前面部分的习惯而来，目的是创造出更具身临其境的宽幅图像和视频，而无需使用拍摄 360 度视频的繁重设备。Mirage 相机是第一批 VR180 相机之一，契合 Google 捕捉 180 度全景图像和视频的新方式。双镜头可拍摄 4K 视频，用户可以在任何 VR 设备上观看三维视频，以及 YouTube 和 Google 照片。

b) Pico Neo 6Dof

图2.9 Pico Neo 6Dof设备

- 双眼分辨率：2880x1600

- 视场角：101°
- 无需视力调节，自适应瞳距
- 双目摄像头，用于头部6DoF空间定位
- 超声波定位传感器，用于手部6DoF空间定位
- 关键传感器：高精度九轴传感器

c) Vive-Focus

图2.10 Vive-Focus设备

- World-Scale 六自由度大空间追踪技术，高精度九轴传感器，距离传感器
- 刷新率 75Hz
- 视场角 110 度
- 搭配高精度九轴传感器操作手柄，更具交互性

浅析 Oculus 的成功

具有深度信息且持续的 3D 渲染是虚拟现实最重要的部分。为了达到这个效果，VR 设备都需要借助一个立体显示设备（即 HMD）。

过去，由于缺乏价格低廉并且长时间佩带舒适的 HMD 设备，以致 VR 头显设备无法大范围进行推广。而 Oculus VR 团队改变了这个局面。他们在 2012 年推出了 Oculus Rift，这款产品具有立体显示功能并且内置头动追踪设备。这款设备不但轻便，而且售价只有几百美元。尽管 Oculus Rift（DK1）的分辨率还很低，但这也足以引起一场产业风暴。新版本的 Oculus Rift（DK2），分辨率、移动追踪性能和显示效果都有所提升。

Oculus Rift 都做了哪些革新性的工作呢？第一，为了产生深度信息，它为每个眼睛生成一张图片，这两张图片在视觉上有一些偏移量，这样就可以模拟人眼的视差。第二，为了产生更好的视觉效果，它通过桶形畸变技术，将图片扭曲从而模拟人眼的球形表面。

有了以上这些技术，Tuscany VR Demo 场景在虚拟现实设备中显示如图 2.11。

图 2.11 Oculus Rift 连接电脑以后通过电脑屏幕看到头显中的真实图像

2.2 渲染相关技术

2.2.1 渲染的定义

渲染过程试图利用有限数目的像素将图像空间连续函数表现成颜色，在电脑绘图中是指用软件从模型生成图像的过程。模型是用严格定义的语言或者数据结构对于三维物体的描述，它包括几何、视点、纹理以及照明信息。

假设我们在一个空的三维空间中要确定一个面，那么至少得有三个点，而一个物体就是由一个个的面构成，在图形学中一般用三角面代替。这是在我们假象的三维空间中，但是如果要在二维的显示器中看到该物体，便需要将该物体的三角面从三维空间渲染到二维空间中，如图 2.12 所示。

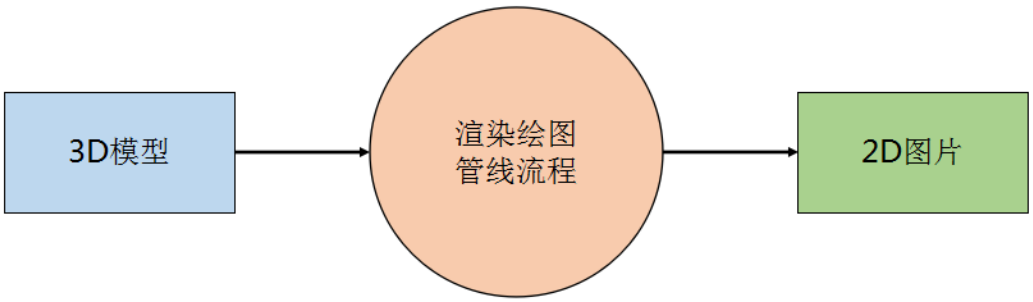

图 2.12 渲染图示：从 3D 到 2D

几何模型

我们首先需要一个虚拟世界来包含几何模型。为此，一个具有笛卡尔坐标的 3D 欧几里德空间就足够了。接着，令 R3 表示虚拟世界，其中每个点都表示为一个三元组实值坐标：（x，y，z）。虚拟世界的坐标系如图 2.13 所示。

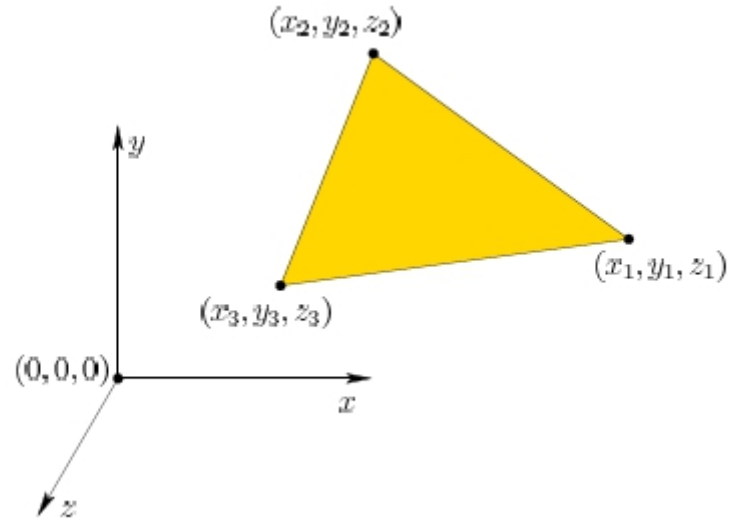

图 2.13 虚拟世界的几何模型

几何模型由 R3 中的曲面或实心区域组成，并包含无限多点。由于计算机中的表示是有限的，模型是根据基元来定义的，其中每个基元表示一组无限点。最简单和最有用的基元是一个 3D 三角形，如图 2.13 中的三角形。对应于“内部”所有点和三角形边界上的平面由三角形顶点的坐标完全指定。为了模拟虚拟世界中的复杂物体，我们可以将大量的三角形排列成一个网格，如图 2.14 所示。

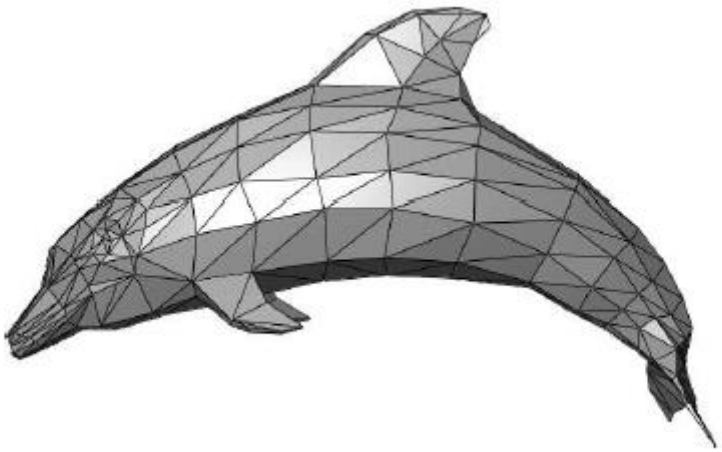

图 2.14：一只海豚的几何模型，由 3D 三角形的网格组成。（来自维基百科用户 Chrschn）

为什么是三角形

使用三角形是因为它们对于算法来说是最方便处理的，特别在硬件实施方面。GPU 的实现倾向于较小的表示，以便可以并行地将多个指令的紧凑列表应用于多个模型部分。当然，也可以使用更复杂的原型，如四边形，样条线和半代数曲面。这可能导致更小的模型尺寸，但通常会因处理这样的原型而带来更大的计算开销。例如，相比于两个 3D 三角形来说两个样条曲面很难确定是否碰撞。

物体顺序 vs 图像顺序

假设一个虚拟世界已经以三角形基元的形式被定义。在此基础上，便可以放置一双虚拟

的眼睛，并从一些特定的位置和方向观看虚拟世界。每个三角形都被正确地放置在虚拟屏幕上。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。在该过程中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。更一般的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。渲染方法的主要区别就是如何处理可视性。

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，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将通过虚拟环境被追踪。

2.2.2 计算机图形学中的渲染方法

为了从 3D 场景转换到 2D，场景中的所有物体都需要转换到几个空间。每个空间都有自己的坐标系。这些转换是通过一个空间的顶点转换到另一个空间的顶点来实现的。

光照（lighting），是这个阶段的另一个主要部分，是使用物体表面的法向量来计算的。通过摄像机的位置和光源的位置，可以计算出给定顶点的光照属性。

对于坐标系变换，我们从物体坐标系开始，每个物体都有自己的坐标系，这有利于几何变换，如平移，旋转和缩放。我们进入到世界坐标系，场景中的所有物体都具有统一的坐标系。下一步是转换到视图空间，即摄像机坐标系。想象一下：先在世界空间中放一个虚拟摄像机，然后进行坐标变换，使得摄像机位于视图空间的原点，镜头对准 z 轴的方向。现在我们定义一个所谓的视体（view frustrum），它用来决定我们通过虚拟的 3D 摄像机所能看到的场景，只需要把这些内容渲染出来即可,如图 2.15 所示。

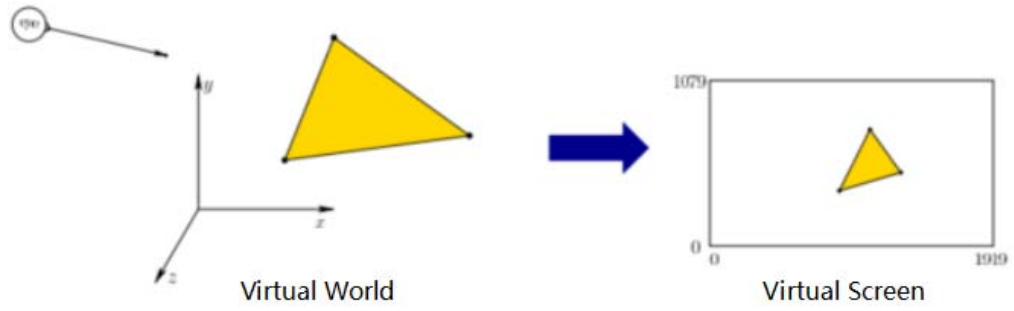

图 2.15 视角坐标变换

光栅化渲染方法

在现实世界中，三角形会将光线向各个方向散射。但是对于一台普通电脑，根本不可能去计算所有不同方向的光线，所以，计算机渲染图像，仅仅计算了那些散射到我们人眼方向的光线，CG 里面也叫摄像机方向，图形学中称之为视角方向。在渲染中，我们需要精确地知道一个光线是照射到一个指定的几何体上的,在该过程中也需要知道射入点的一些几何属性，比如面法线(surface normal)或者它的材质(material)。大部分光线追踪器都包含测量

一个光线和多个物体的相交性，返回距离最近的对象之类的功能。一个光线追踪器应该对场景中的光照进行建模，除了描述光源的位置，也包括描述这些光的能量是如何分布在场景中的。因而假设在 3D 空间中添加一个摄像头，并在前端透视点上放置一个屏幕网格，其中每个框是渲染图像的一个像素。我们只需画出与虚拟相机透视点相交的一些光线。如果这些光线相交于我们的屏幕，那么屏幕就能知道我们观察该三角形的边界位置，如图 2.16 所示。

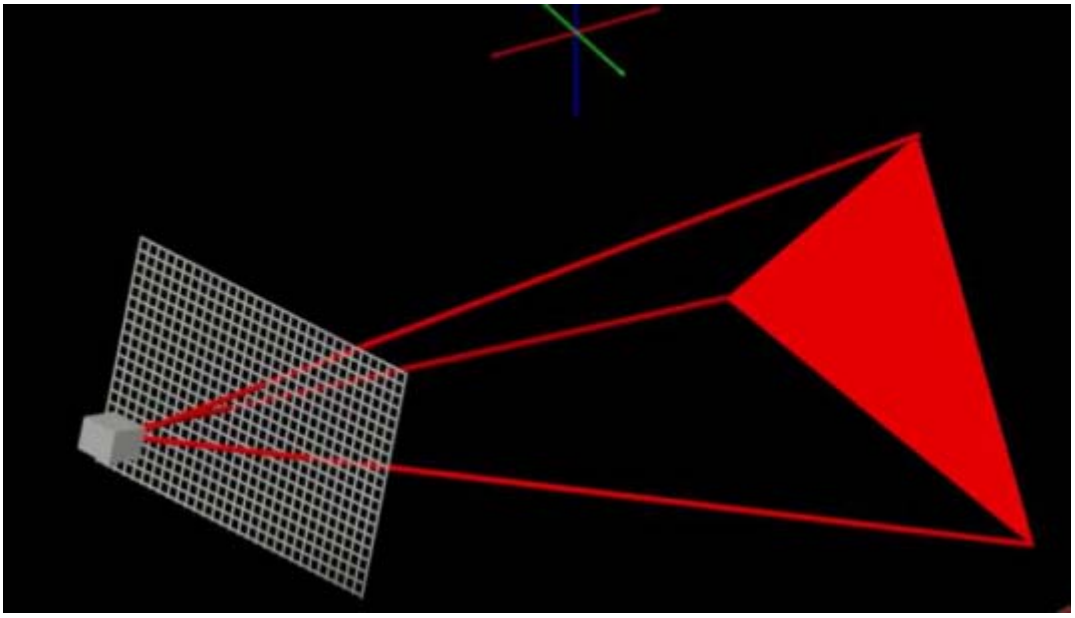

图 2.16 视角方向建模

知道边界以后，对边界与像素之间重叠的部分渲染像素，其余地方不做处理，便可以得到这个三角形的图像，如图 2.17 所示：

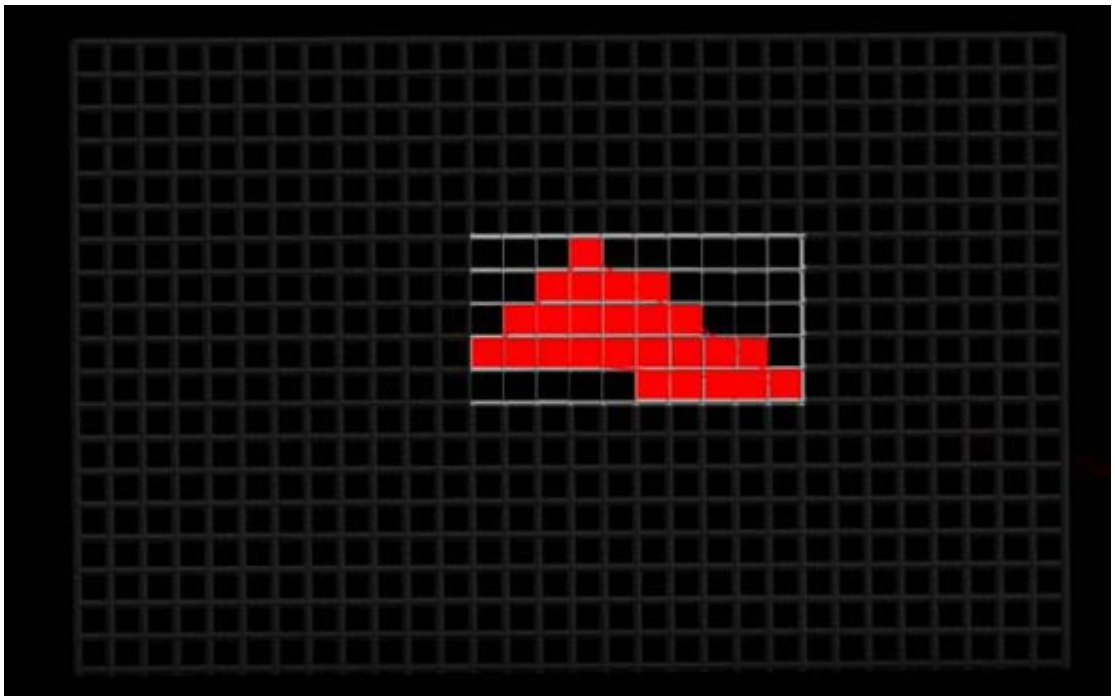

图 2.17 渲染成功的三角形

光线投射

光栅化是以物体为中心的，从相机中捕捉物体所发射到相机的光线，而 Ray casting 以

图像为中心，只考虑那些实际有用的光线，从虚拟相机开始，并通过相机向每一个像素都发射一条光线：

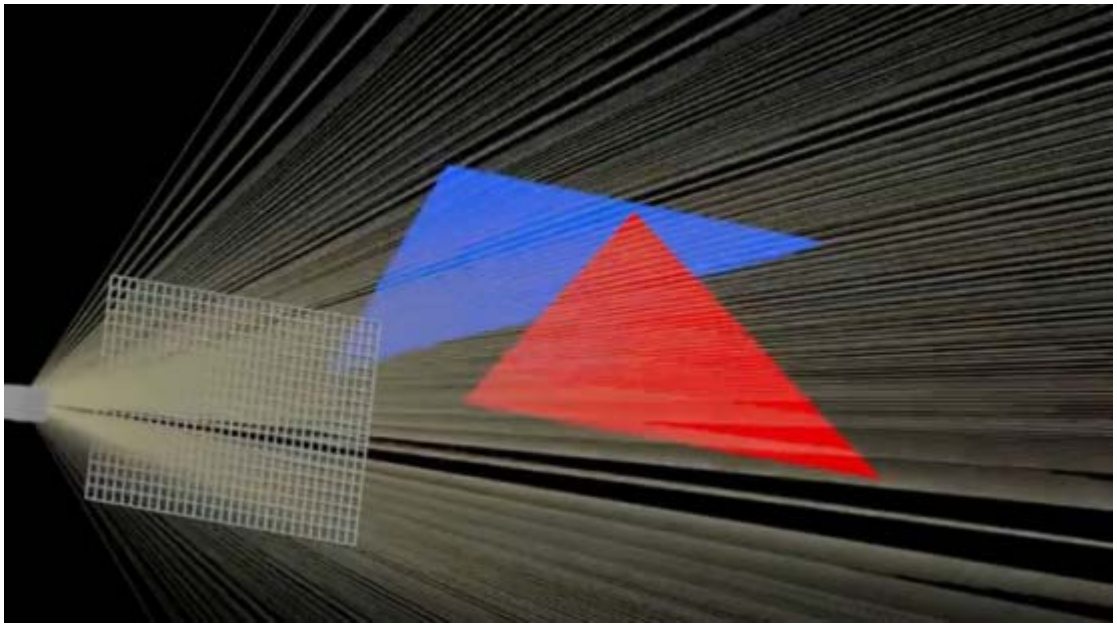

图 2.18 光线投射示意图

现在将判断每条射线是否射中了每一个三角形 ，如果一个光线遇到多个物体，取最近的那个点。如果忽略计算性能，计算观察光线在离开图像像素后碰到的第一个三角形非常简单直接的。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。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，最近的交点就被找到。虽然这种计算很简单，但是它也有很大的弊端，就是计算量过于庞大，因为该方法需要判断相当多的射线与物体的三角面是否相交。假设有一个 1000 x 1000 像素的图像，那么该方法就要计算 1,000,000 条光线是否和场景中一个多边形相交，对计算机来说，这计算相当地费时费力，即使目前在算法上有了很大的改进，仍然需要大量的计算。该方法比较常见的例子包括 BSP 树和 Bounding Volume Hierarchies 。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。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。

光线追踪

光线追踪的流行来源于它比其它渲染方法如扫描线渲染或者光线投射更加能够现实地模拟光线，像反射和阴影这样的一些对于其它算法来说都很难实现的效果，却是光线追踪算法的一种自然结果。光线追踪易于实现并且视觉效果很好。

当主光线接触一个表面时，通过在反射方向上绘制二次光线来绘制阴影光线，只要这个光线可以反射到光源，那么我们就知道是光源照亮了这个物体，如图 2.19 所示。如果我们发现光与表面之间存在物体，这时表面就处于阴影中，如图 2.20 所示，如此不断的反射被称之为递归射线追踪。

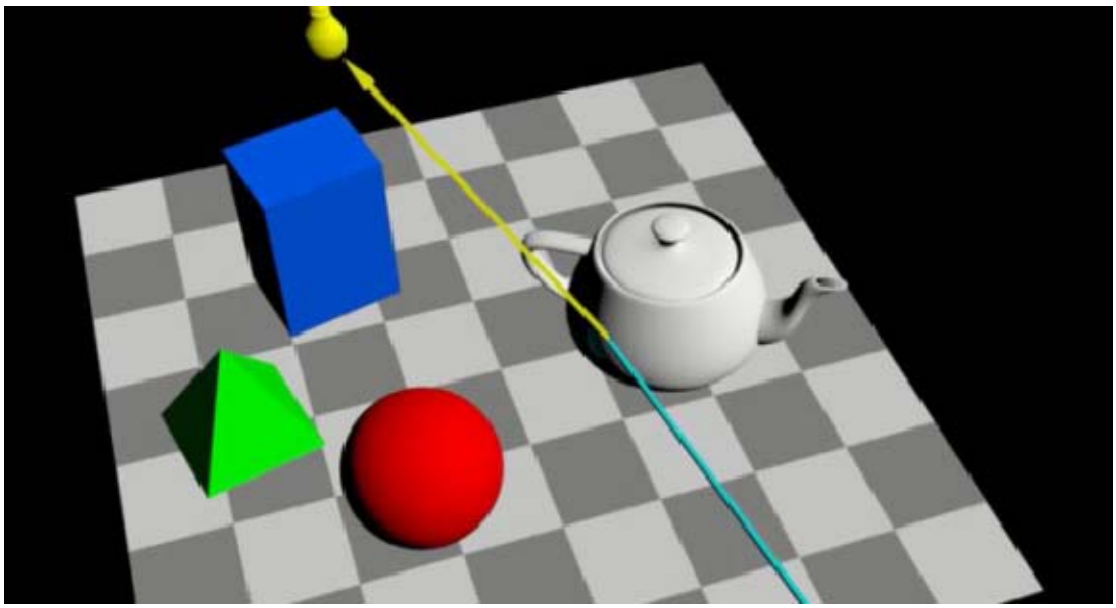

图 2.19 光线追踪绘制反射光线

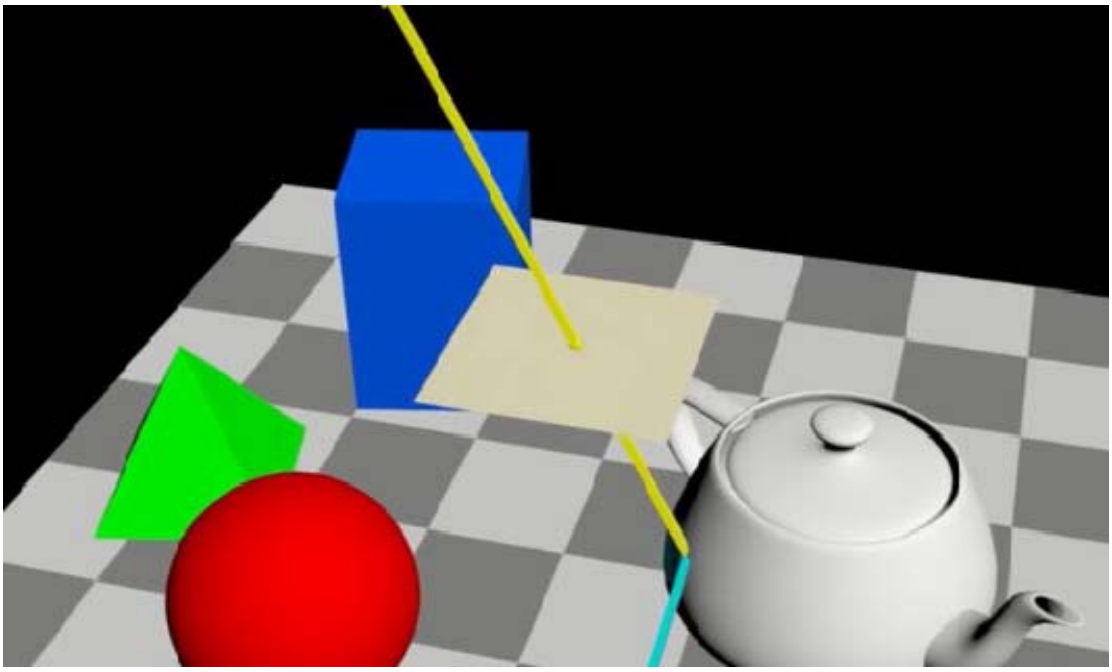

图 2.20 光与表面之间存在物体，至于阴影处

为了计算像素的 RGB 值，根据放置在虚拟世界屏幕上的焦点通过像素的中心绘制观察光线，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。光线追踪的一个最大的缺点就是性能，扫描线算法以及其它算法利用了数据的一致性从而在像素之间共享计算，但是光线追踪通常是将每条光线当作独立的光线，每次都要重新计算。但是，这种独立的做法也有一些其它的优点，例如可以使用更多的光线以抗混叠现象，并且在需要的时候可以提高图像质量。尽管它正确地处理了相互反射的现象以及折射等光学效果，但是传统的光线追踪并不一定是真实效果图像，只有在非常近似或者完全实现渲染方程的时候才能实现真正的真实效果图

像。

2.2.3 光栅化

经过变换的顶点流按照顺序被送到下一个被称为图元装配和光栅化的阶段。首先，在图元装配阶段根据伴随顶点序列的几何图元分类信息把顶点装配成几何图元。这将产生一序列的三角形、线段和点。这些图元需要经过裁剪到可视平截体（三维空间中一个可见的区域）和任何有效的应用程序指定的裁剪平面。光栅器还可以根据多边形的朝前或朝后来丢弃一些多边形。这个过程被称为挑选（culling）。

经过裁剪和挑选剩下的多边形必须被光栅化。光栅化是一个决定哪些像素被几何图元覆盖的过程。多边形、线段和点根据为每种图元指定的规则分别被光栅化。光栅化的结果是像素位置的集合和片段的集合。当光栅化后，一个图元拥有的顶点数目和产生的片段之间没有任何关系。例如，一个由三个顶点组成的三角形占据整个屏幕，因此需要生成上百万的片段。片段和像素之间的区别变得非常重要。术语像素（Pixel）是图像元素的简称。一个像素代表帧缓存中某个指定位置的内容，例如颜色，深度和其它与这个位置相关联的值。一个片段（Fragment）是更新一个特定像素潜在需要的一个状态。之所以术语片段是因为光栅化会把每个几何图元（例如三角形）所覆盖的像素分解成像素大小的片段。一个片段有一个与之相关联的像素位置、深度值和经过插值的参数，例如颜色，第二（反射）颜色和一个或多个纹理坐标集。这些各种各样的经过插值的参数是来自变换过的顶点，这些顶点组成了某个用来生成片段的几何图元。你可以把片段看成是潜在的像素。如果一个片段通过了各种各样的光栅化测试，这个片段将被用于更新帧缓存中的像素。

光栅操作阶段根据许多测试来检查每个片段，这些测试包括剪切、alpha、模板和深度等测试。这些测试涉及了片段最后的颜色或深度，像素的位置和一些像素值（像素的深度值和模板值）。如果任何一项测试失败了，片段就会在这个阶段被丢弃，而更新像素的颜色值（虽然一个模板写入的操作也许会发生）。通过了深度测试就可以用片段的深度值代替像素深度值了。在这些测试之后，一个混合操作将把片段的最后颜色和对应像素的颜色结合在一起。最后，一个帧缓存写操作用混合的颜色代替像素的颜色。

2.2.4 贴图

当一个图元被光栅化为一个或多个片段时，插值、贴图和着色阶段就在片段属性需要的时候插值，执行一系列的贴图和数学操作，然后为每个片段确定一个最终的颜色。除了确定片段的最终颜色，这个阶段还确定一个新的深度，或者甚至丢弃这个片段以避免更新帧缓存对应的像素。

纹理贴图

在计算机图形学中，纹理贴图是使用图像、函数或其他数据源来改变物体表面外观的技术。例如，可以将一幅砖墙的彩色图像应用到一个多边形上，而不用对砖墙的几何形状进行精确表示。当观察这个多边形的时候，这张彩色图像就出现在多边形所在位置上。只要观察者不接近这面墙，就不会注意到其中几何细节的不足（比如其实砖块和砂浆的图像是显示在光滑的表面上的事实）。通过这种方式将图像和物体表面结合起来，可以在建模、存储空间和速度方面节省很多资源。

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，如图 2.21 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。

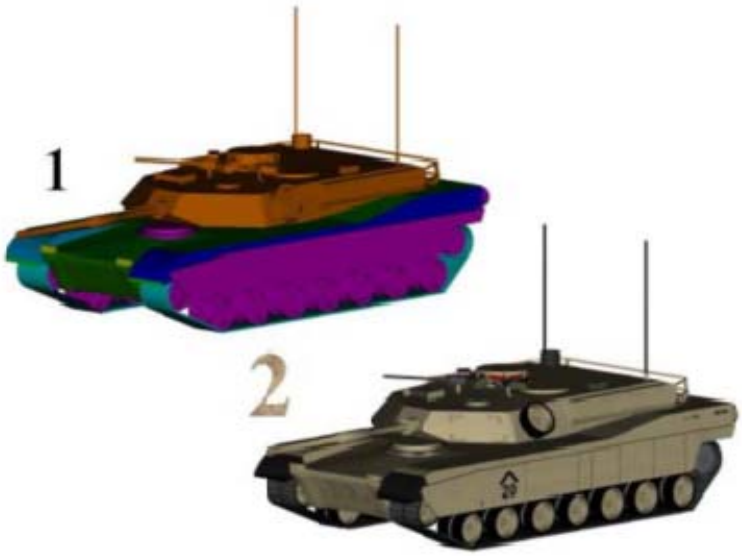

图 2.21 纹理贴图：将简单的图案或是整幅图像映射在三角形纹理中，然后在图像上进行渲染，相比于直接使用模型中的三角形纹理，这样可以提供更多细节。（图源于 Wikipedia）

纹理管线

简单来说，纹理（Texturing）是一种针对物体表面属性进行“建模”的高效技术。图像纹理中的像素通常被称为纹素（Texels），区别于屏幕上的像素。根据 Kershaw 的术语，通过将投影方程（projector function）运用于空间中的点，可以得到一组称为参数空间值（parameter-spacevalues）的关于纹理的数值。这个过程就称为贴图（Mapping，也称映射），也就是纹理贴图（Texture Mapping，也称纹理映射）这个词的由来。纹理贴图可以用一个通用的纹理管线来进行描述。纹理贴图过程的初始点是空间中的一个位置。这个位置可以基于世界空间，但是更常见的是基于模型空间。因为若此位置是基于模型空间的，当模型移动时，其纹理才会随之移动。如图 2.22 为一个纹理管线（The Texturing Pipeline），也就是单个纹理应用纹理贴图的详细过程，而此管线有点复杂的原因是每一步均为用户提供了有效的控制。

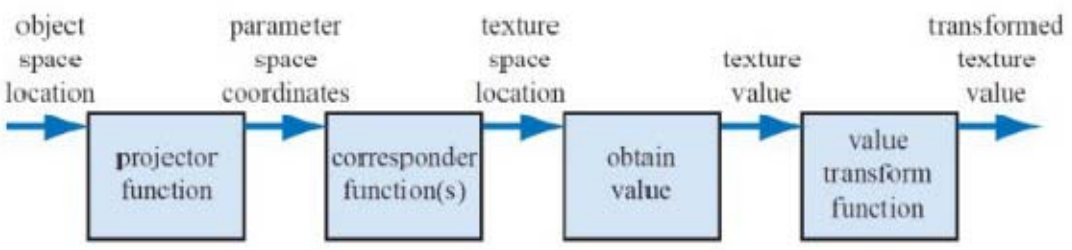

图 2.22 纹理管线流程图

第一步。通过将投影方程（projector function）运用于空间中的点 ，从而得到一组称为参数空间值（parameter-space values）的关于纹理的数值。

第二步。在使用这些新值访问纹理之前，可以使用一个或者多个映射函数（corresponder function）将参数空间值（parameter-space values ）转换到纹理空间。

第三步。使用这些纹理空间值（texture-space locations）从纹理中获取相应的值（obtain value）。例如，可以使用图像纹理的数组索引来检索像素值。

第四步。再使用值变换函数（value transform function）对检索结果进行值变换，最后使用得到的新值来改变表面属性，如材质或者着色法线等等。

投影函数 The Projector Function

作为纹理管线的第一步，投影函数的功能就是将空间中的三维点转化为纹理坐标，也就是获取表面的位置并将其投影到参数空间。在常规情况下，投影函数通常在美术建模阶段使用，并将投影结果存储于顶点数据中。也就是说，在软件开发过程中，我们一般不会去用投影函数去计算得到投影结果，而是直接使用在美术建模过程中，已经存储在模型顶点数据中的投影结果。通常在建模中使用的投影函数有球形、圆柱、以及平面投影，也可以选其他一些输入作为投影函数。

映射函数 The Corresponder Function

映射函数（The Corresponder Function）的作用是将参数空间坐标（parameter-space coordinates）转换为纹理空间位置（texture space locations）。我们知道图像会出现在物体表面的(u,v)位置上，且 u，v 值的正常范围在[0,1)范围内。超出这个值域的纹理，其显示方式便可以由映射函数（The Corresponder Function）来决定。

法线贴图

另一种可行方法是法线贴图，是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。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。

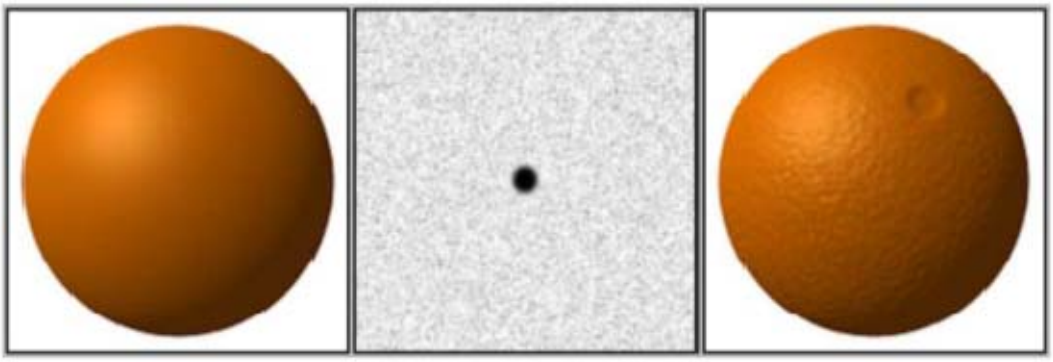

图 2.23 凹凸贴图：通过人工改变表面法线，阴影生成算法会产生一个看上去粗糙的表面（图源于 Brian Vibber）

凹凸贴图与其改进

凹凸贴图是指计算机图形学中在三维环境中通过纹理方法来产生表面凹凸不平的视觉效果。它主要的原理是通过改变表面光照方程的法线，而不是表面的几何法线，或对每个待渲染的像素在计算照明之前都加上一个从高度图中找到的扰动，来模拟凹凸不平的视觉特征，如褶皱、波浪等等。Blinn 于 1978 年提出了凹凸贴图方法。使用凹凸贴图，是为了给光滑的平面，在不增加顶点的情况下，增加一些凹凸的变化。该方法的原理是通过法向量的变化，来产生光影的变化，从而产生凹凸感。实际上并没有顶点（即 Geometry）的变化。

以下是几种凹凸贴图与其改进方法的总结对比，如图 2.24 所示。

贴图方式	思想概述	提出年代
Bump mapping 凹凸贴图	计算 vertex 的光强时，不是直接使用该 vertex 的原始法向量，而是在原始法向量上加上一个扰动得到修改法向量，经过光强计算，能够产生凹凸不平的表面效果。No self-occlusion, No self-shadow, No silhouette。	1978
Displacement Mapping 移位贴图	直接作用于 vertex，根据 displacement map 中相对应 vertex 的像素值，使 vertex 沿法向移动，产生真正的凹凸表面。	1984
Normal Mapping 法线贴图	normal map 需要法向量的信息，而法向量信息可由 height map 得到，且 texture 的 RGB 可以表示法向量的 XYZ，利用此信息计算光强，产生凹凸阴影的效果。No self-occlusion, No self-shadow, No silhouette。	1996
Parallax Mapping (Virtual Displacement Mapping) 视差贴图	没有修改 vertex 的位置，以视线和 height map 计算较陡峭的视角给 vertex 较多的位移，较平缓的视角给 vertex 较少的位移，透过视差获得更强的立体感，即利用 HeightMap 进行了近似的 Texture Offset。No self-occlusion, No self-shadow。	2001
Relief Mapping (Steep Parallax Mapping) 浮雕贴图	更精确地找出观察者的视线与高度的交点，对应的 texture 坐标则是位移的距离，所以能更正确地模拟立体的效果。Relief Mapping 实现了精确的 Texture Offset。Relief Mapping 可以产生 self-occlusion, self-shadowing, view-motion parallax, and silhouettes。	2005

图 2.24 凹凸贴图与其改进方法的总结对比

表示凹凸效果的另一种方法是使用高度图来修改表面法线的方向。每个单色纹理值代表一个高度，所以在纹理中，白色表示高高度区域，黑色是低高度的区域（反之亦然）。示例如图 2.25。

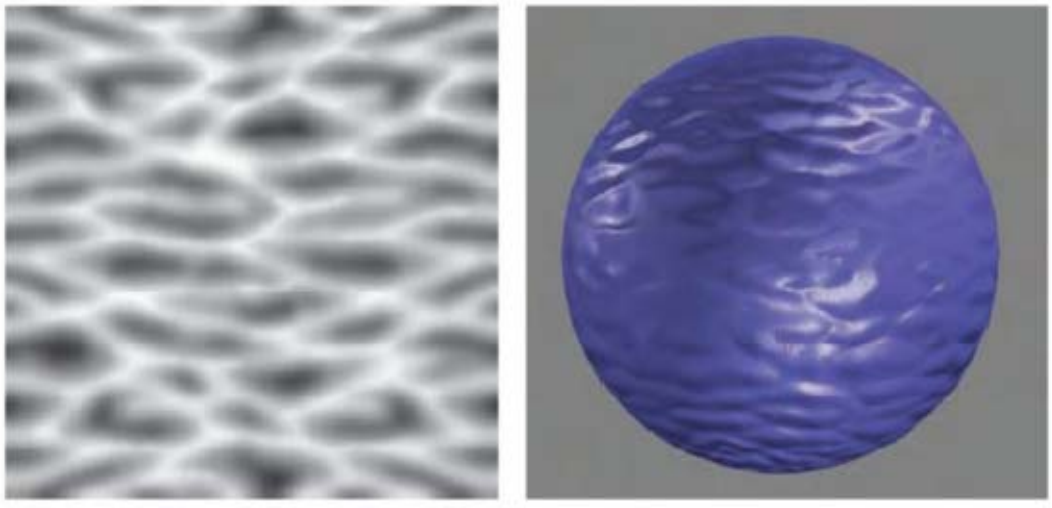

图 2.25 使用高度图来修改表面法线的方向造成凹凸不平的视觉效果

2.3 光学相关技术（反畸变）

2.3.1 光的基本行为

光可以用三种看起来相互矛盾的方式来描述：

1. 光子：在空间中高速移动的微小能量粒子。当考虑传感器或接收器接收的光子数量时，这种解释是有帮助的。

2. 光波：通过空间的波纹，类似于在水面上传播的波浪，但是是三维的。波长是峰值之间的距离。在研究颜色的光谱时，这种解释是有帮助的。

3. 光线：光线追踪单个假想光子的运动。方向垂直于波前（见图 2.26）。这个解释在解释镜头和定义可见性概念时很有用。

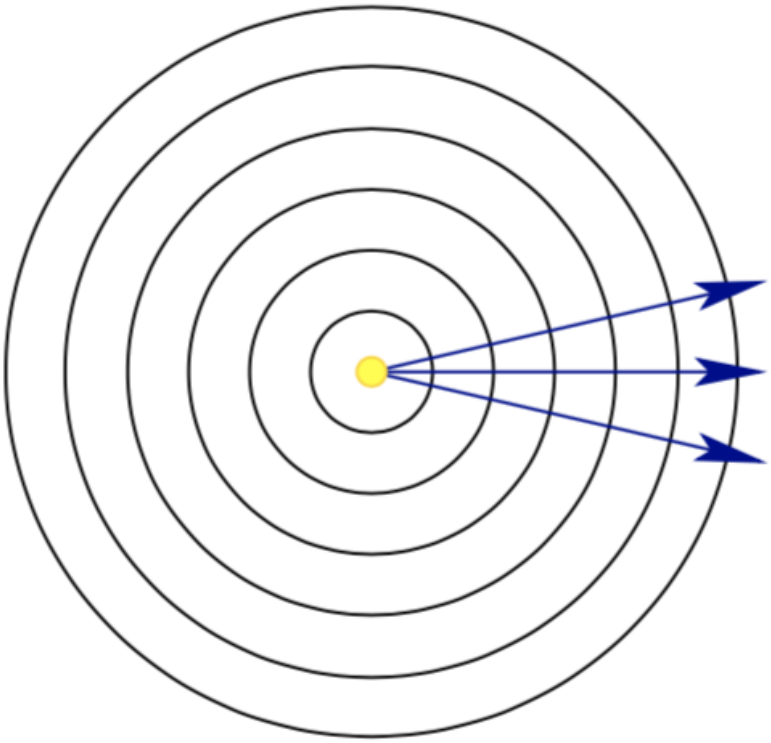

图 2.26 从点光源发出的波和可见光线

与材料的相互作用

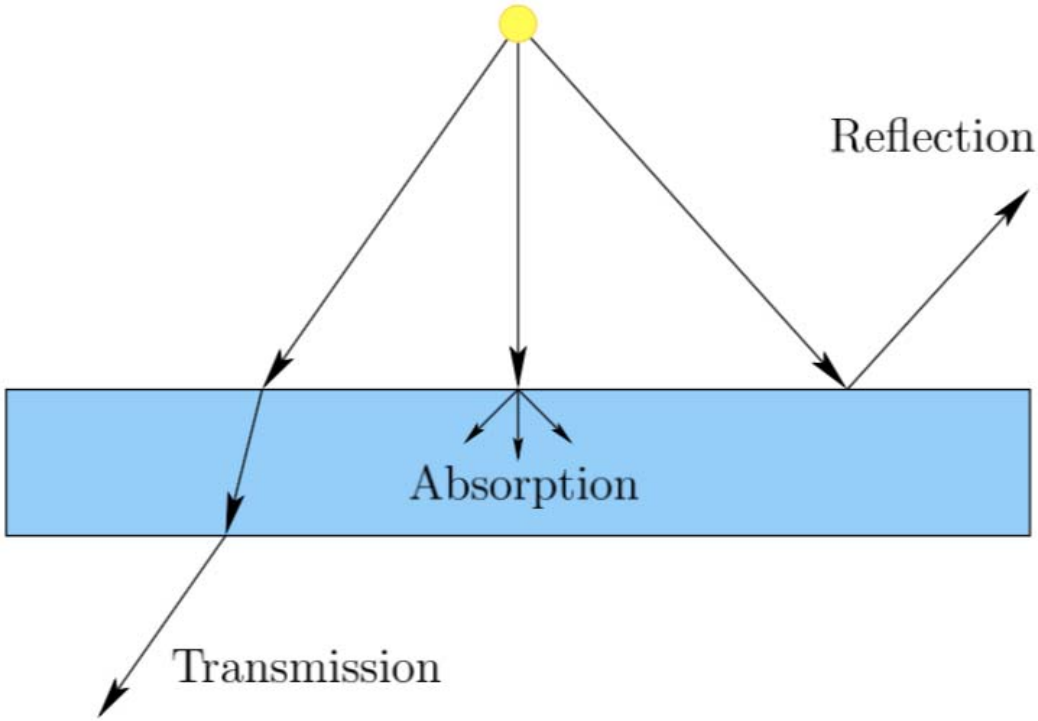

图 2.27 当光能碰到不同介质的边界时，有 3 种可能性：透射，吸收和反射

当光线撞击材料表面时，可能会发生三种行为之一，如图 2.27 所示。在透射的情况下，能量穿过材料并从另一侧离开。对于像玻璃这样的透明材料，透射光线会按照斯涅尔定律减速并弯曲。对于不透明的半透明材料，射线在离开之前会散射到各个方向。在吸收的情况下，当光被捕获时，能量被材料吸收。第三种情况是反射，其中光线从表面偏转。沿着完美光滑或抛光的表面，光线以相同的方式反射：出射角等于入射角。这种情况称为镜面反射，与漫反射相反。反射光线在任意方向上散射。通常，所有三种传播，吸收和反射的情况同时发生。这些情况取决于许多因素，例如接近角度，波长以及两种相邻材料或介质之间的差异。光被不同地模拟为几何光线，电磁波或光子（具有一些波特性的量子粒子）。无论怎样处理，光都是通过空间传播的电磁能。根据渲染目的，光源可以以许多不同的方式来表示。光源可以分为三种不同类型：平行光源、点光源和聚光灯。

2.3.2 光学畸变

沉浸感需要大的视场角，可以通过将一个大的弯曲的球形显示器放到面前的方式来实现，但是这样的方案是非常昂贵的，一个更加实惠的解决方案是通过在一个小的矩形显示屏上增加一个透镜，然后通过透镜来观看显示屏，从而获得更大的视场角，如图 2.28 所示：

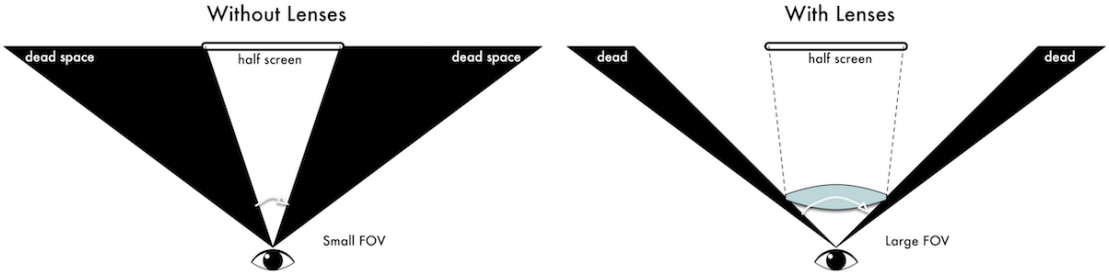

图 2.28 通过添加一个透镜扩大视野范围

虽然放置在眼睛附近的镜片会大大增加视野，但与此同时会付出一定的代价：图像产生

球形失真。视野越大，图像越扭曲。许多统称为畸变的缺陷会降低由镜头形成的图像质量。由于这些问题在日常使用中都很明显，需要采取相关的补偿措施并应用到 VR 系统中。

色差

光通常是一束具有波长光谱的波。当白光通过棱镜折射时，整个可见光谱按颜色被很好地分开。这是一个美丽的光学现象，但对于镜头来说，这是非常糟糕的，因为它分散了图像的各种颜色成分。这个问题被称为色差。

问题的实质在于通过介质的光速取决于波长，图 2.29 展示了简单凸透镜的色差现象。此时，焦距为波长的函数。如果我们沿着相同的光线将红色，绿色和蓝色激光直接照射到镜头中，则每种颜色的光线会在不同的位置穿过光轴，产生红色，绿色和蓝色焦点。对于常见的光源和介质，透过镜头的光线会形成连续的焦点集合。图 2.30 显示了一个具有色差伪像的图像。我们可以通过组合不同介质的凸透镜和凹透镜减少色差，使得发散的射线被强制收敛。

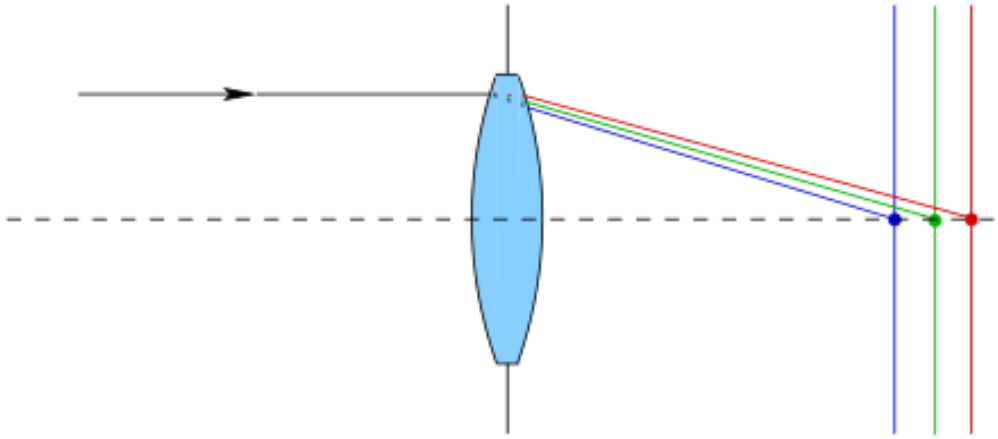

图 2.29 色差是因不同波长的光线在同一介质内的不同光速所引起的，导致每种颜色的光线都有不同的聚焦平面。

图 2.30 上子图被适当地聚焦，而下子图遭受了色差问题。（图由 Stan Zurek 提供）

像散性

图 2.31 描述了像散性，这是对不垂直于晶状体的入射光线发生的晶状体像差。直到现在，我们的镜头图纸都是 2D 的，然而，需要引入第三个维度来了解这种新的畸变。射线可以在一维上发生轴偏移，但在另一维中对齐。通过沿着光轴移动图像平面，不可能使图像成为焦点。相反，这样会出现水平和垂直震源深度，如图 2.31 所示。

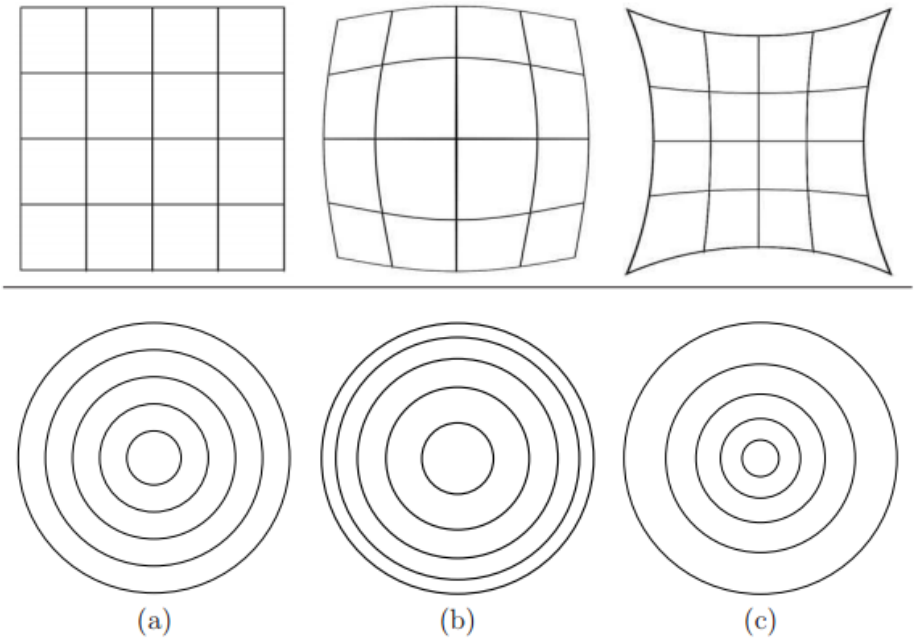

图 2.31 常见的光学畸变。（a）原始图像（b）桶形失真（c）枕形失真。对于第一行的畸变，网格变得非线性扭曲。第二行说明它仍然保持圆对称。

图 2.32 一个带有桶形失真的图像，由鱼眼镜头拍摄。（图片由维基百科用户 Ilveon 提供）

光学畸变的修正

对于大视场的光学系统，桶形畸变和枕形畸变是很常见的（如图 2.32、2.33）。通过 VR 头显镜头观看时，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，这也有助于研究导致虚拟现实疾病的成因，可能是由于在 VR 体验时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许多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可以直接将光投射到眼睛中。

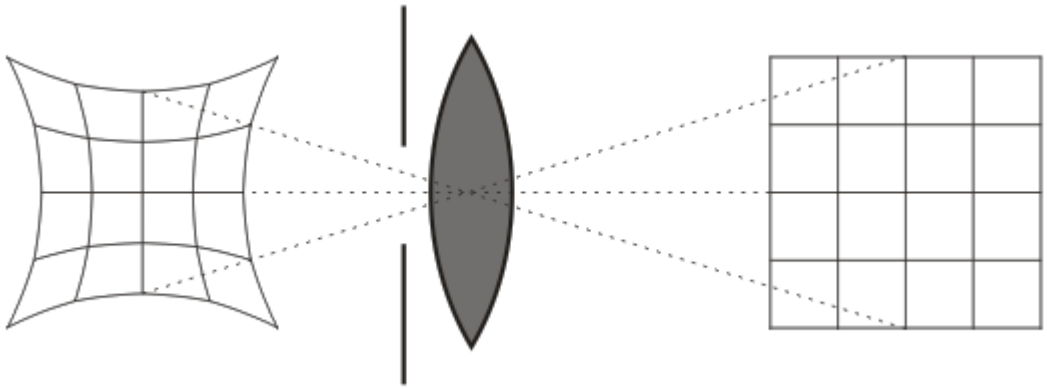

图 2.33 头戴式显示器的镜头枕形失真

解决方法是对这些畸变的图像使用”桶形”畸变，如图 2.34 所示，当我们通过畸变透镜上看,这些经过反畸变的图像看起来就是正常的：

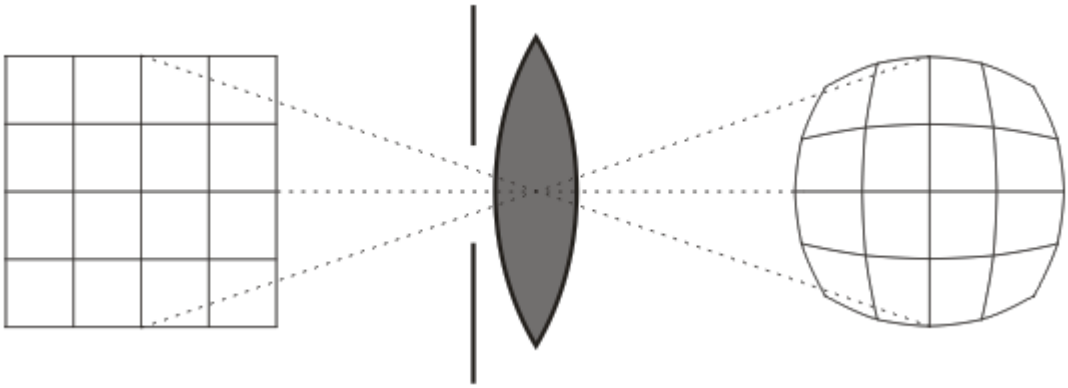

图 2.34 对图像提前进行桶形畸变，使得图像正常

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。

基于像素的处理

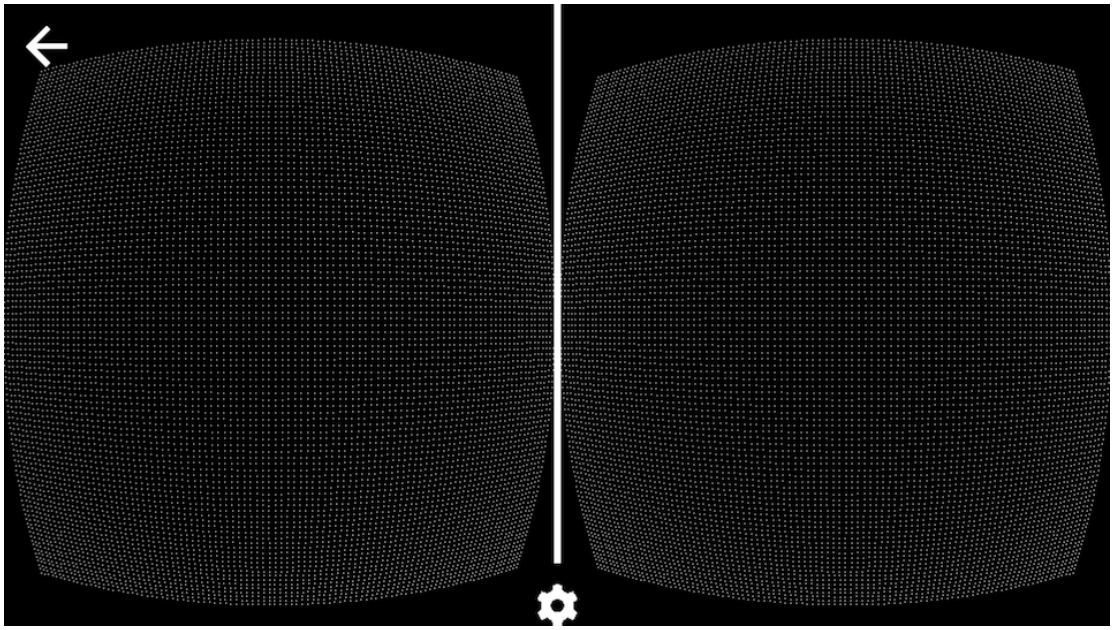

图 2.35 对所有像素进行处理

透镜失真在数学上是很好理解的，由方程控制，失真系数对应于特定透镜。要正确地消除失真，我们还需要计算眼睛的中心，这就需要了解显示器的几何形状和外壳本身。在此方法中我们单独处理每个像素,对每个像素进行”桶型”畸变的数学变换。首先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等，然后取最佳的拟合。其次确定 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 的逆函数可以抵消畸变带来的影响。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函数进行拟合。

基于网格的处理

与基于像素的处理方法不同，这里并不是单独处理每个像素，而是扭曲相对稀疏网格的顶点，如图 2.36 所示。

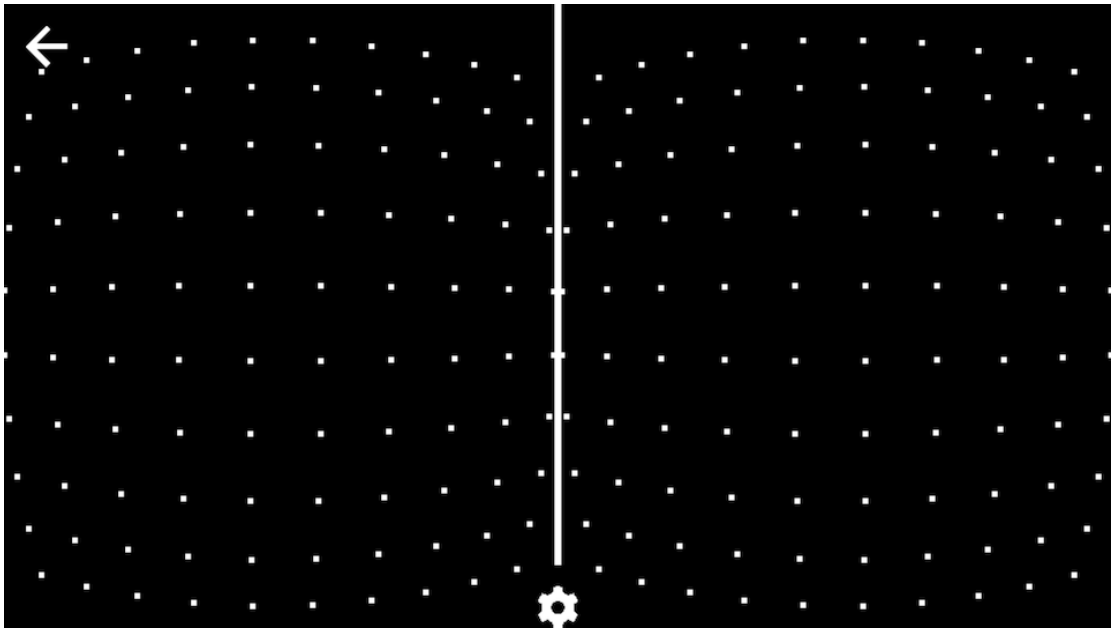

图 2.36 基于网格的处理

对网格中的每个顶点进行计算可以节省一些直接计算量，计算步骤显著减少，并且性能得到了很好的提升。

2.3.3 异步时间扭曲

异步时间扭曲（Asynchronous Timewarp 简称 ATW）是一种生成中间帧的技术，当游戏不能保持足够帧率时，ATW 能产生中间帧，从而有效减少游戏画面的抖动。

大部分的显示器和绝大部分的手机屏幕的刷新率都是 60Hz，也就是说，在理想情况下我们的显示设备大概要在每秒处理 60 帧的画面，也就是说从数据到渲染就有 1000/60≈16.6666ms 的时间延迟。对于虚拟现实设备，为了要在虚拟世界里呈现给人们正确的感知图像，必须要在显示器上定时更新图像，然而，如果渲染时间太长，对应帧就会丢失，产生的结果就是抖动，帧率不稳定。那么，如何抵消这个时延呢？John Carmack 提出一种方法：通过大量采集陀螺仪数据，在样本足够多的情况下，就可以预测出 16.67ms 后用户头部应有的旋转和位置，按照这个预测的数据来渲染。时间扭曲则是一种图像帧修正的技术，它通过扭曲一幅将被送往显示器的图像，来解决这个 16.67ms 的延迟。最基础的时间扭曲是基于方向的扭曲，这种只纠正了头部的转动变化姿势，这种扭曲对于 2D 图像是有优势的，它可以合并一幅变形图像且不需要花费太多系统资源。

异步时间扭曲是指在一个线程（称为 ATW 线程）中进行处理，这个线程和渲染线程平行运行（异步），如果没有时间扭曲，HMD 将捕获有关头部位置的数据，根据此数据渲染图像（正确的角度等），然后在下一个场景到达屏幕时显示图像。在 60 fps 的游戏中，每 16.7 毫秒显示一个新场景。通过此过程，您看到的每个图像都基于近 17 毫秒前的头部跟踪数据。

使用时间扭曲，该过程的前两部分是相同的。HMD 将捕获有关头部位置的数据，并根据数据渲染图像。在显示此图像之前， HMD 会再次捕获头部位置。使用此信息修改渲染图像以适合最新数据。最后，修改后的图像显示在屏幕上。生成的图像更准确地描绘了显示时头部的位置，而不同于最初渲染的图像。时间扭曲仅适用于非常短的时间间隔。

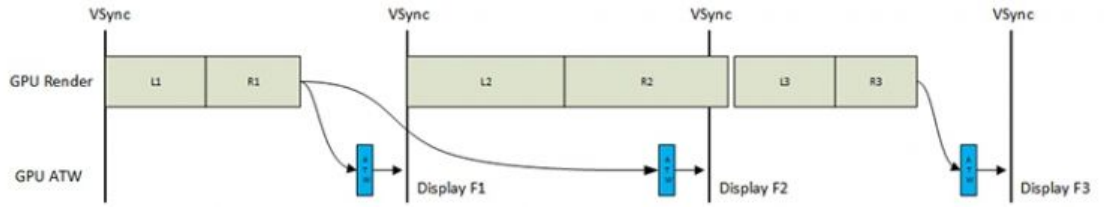

图 2.37 GPU 渲染过程中插入 ATW

GPU 分别为左右眼的画面进行渲染，然后在画面显示出来之前插入一个 ATW 的处理过程。在左边这帧的处理中，画面渲染及时完成，此时进行直接显示，若中间的第二帧渲染未能及时完成，会出现画面抖动。但添加的 ATW 进程会将前面一帧调用出来重新显示，同时加上头盔运动变化，从而保持帧率。

异步时间扭曲局限性

1： 它需要 GPU 硬件支持合理的抢占粒度。在 90Hz 时，帧间间隔大约是 11ms，这意味着为了使 ATW 有机地生成一帧，它必须能够抢占渲染线程并且运行时间少于 11ms，然而 11ms 并不友好，如果 ATW 在一帧时间区间内任意随机点开始运行，那么潜伏期（执行和帧扫描之间的时间）也将随机，并且需要确保不跳跃任何渲染帧。而对现在的图形驱动实现来说，2ms 抢占是一个艰巨的任务。

2： 它要求操作系统和驱动程序支持 GPU 抢占。如果抢占操作不是很快，则 ATW 将无法抢在画面同步之前生成中间帧。这样会使得最后一帧再显示，进而导致抖动，这意味着正确的异步时间扭曲实现应该能够抢占和恢复任意渲染操作和管线状态。理论上讲，目前已有的三角抢占（triangle-granularity）也不够好，因为我们不知道一个复杂着色器执行将花多长时间。

另一方面是操作系统对抢占的支持，在 Windows8 之前，Windiows 显示驱动模型（WDDM）支持使用“批处理队列”粒度的有限抢占。但不幸的是，图形驱动程序趋向于大批量渲染效率，会导致 ATW 的实现过于粗糙。

总体来说 ATW 确实是一项很棒的技术，没有它的话，开发者在游戏开发中为了保持画面帧率只能非常保守地使用 CPU 和 GPU 性能，而 ATW 可以游戏更容易保持帧率稳定，从而让开发者在画面设计上更加大胆。实际运行 Oculus 中可以发现，没有使用 ATW 的 app 在运行中丢失了约 5%的帧。ATW 可以将大部分丢失的帧补上，从而大幅减少画面抖动。而这一切对 app 来说不需要消耗更多性能或更改代码就能实现。Oculus 还表示这一切只是开始，他们正与合作伙伴尝试提高 ATW 的运行效率。

2.4 常用的终端集成引擎

对于使用者来说，关于 VR 技术的最直接感受就是全景视频，也称 360 度全景视频或沉浸式视频。现在有很多终端集成引擎能够实现全景视频的呈现与播放，就目前来说，使用最多的是 Unity3D 软件，Unreal 和 WebVR 等，这些都能够很好的完成全景视频的呈现。

Unity3D

Unity3D 软件是由 Unity Technologies 开发的一个能够创建三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的游戏和相关软件的开发工具。该工具可以发布游戏或者软件至 Windows、Mac、Wii、iPhone、WebGL（需要 HTML5）、Windows phone 和 Android 平台，也可以利用 Unity web player 插件发布网页游戏，支持 Mac 和 Windows 的网页浏览，它的网页播放器也被 Mac 所支持。

下图是 Unity 的初始界面：

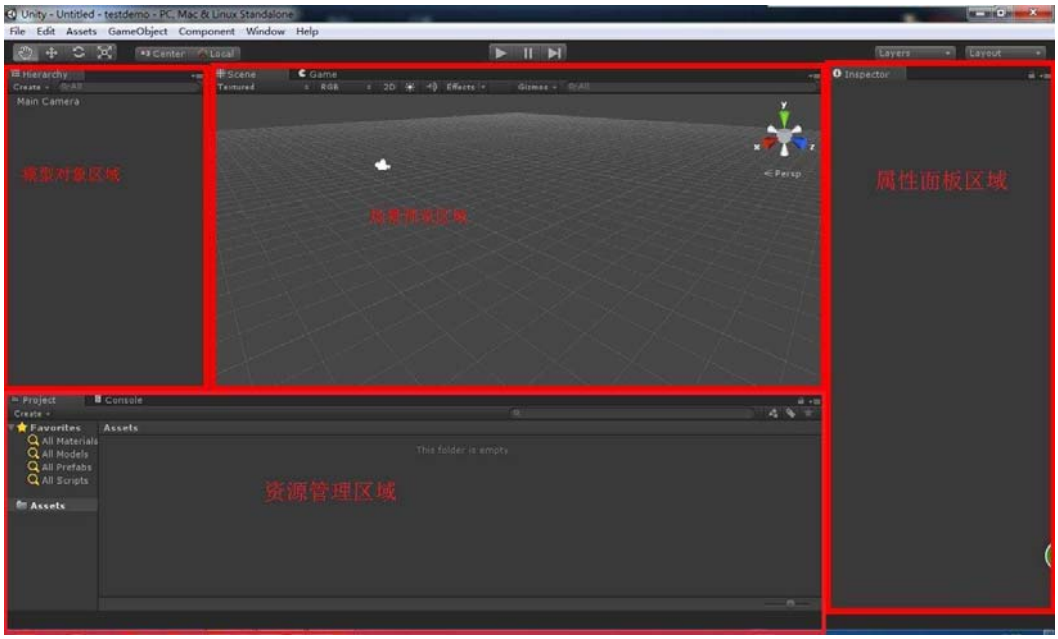

图 2.38 Unity 初始界面

要想很快地完成一个能够实现全景视频呈现的软件，一个比较好的方式就是通过使用现有 Unity 中的 SDK，比如 googlvr for unity，Occulus Unity SDK，AVpro SDK 等，这些 SDK 中都很好的集成了相关功能，方便了开发者的使用。下面介绍如何通过 Unity 和 GoogleVR for Unity 来创建一个简单的全景视频播放器。

首先新建一个 unity 项目，在新建场景中添加一个球体，position 设置为原点(0,0,0)，size 可以设置为(5,5,5)，也可以根据需要设置大小，MainCamera 的 position 同样设置为原点(0,0,0)，这样就相当于观看者站在球心的位置，而相应的球体内部应该播放的是全景视频，Unity 默认是不会将球体的内部渲染出来，所以现在需要通过着色器翻转球体的法线。

新建材质给球体，再将新建的 shader 给新建的材质。Shader 代码如下：

```
Shader "Custom/flipnormal" {
    Properties {
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
    }

    SubShader {
        Tags { "RenderType"="Opaque" }
        Cull Off
        CGPROGRAM
        #pragma surface surf Lambert vertex:vert
        sampler2D _MainTex;
        struct Input {
            float2 uv_MainTex;
            float4 color:COLOR;
        };

        void vert(inout appdata_full v)
        {
            v.normal.xyz=v.normal*-1;
        }
        void surf (Input IN, inout SurfaceOutput o) {
            fixed3 c = tex2D (_MainTex, IN.uv_MainTex);
            o.Albedo = c.rgb;
            o.Alpha = 1;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

接下来为球体添加 VideoPlayer 组件，将准备好的.mp4 全景视频拖至 Video clip

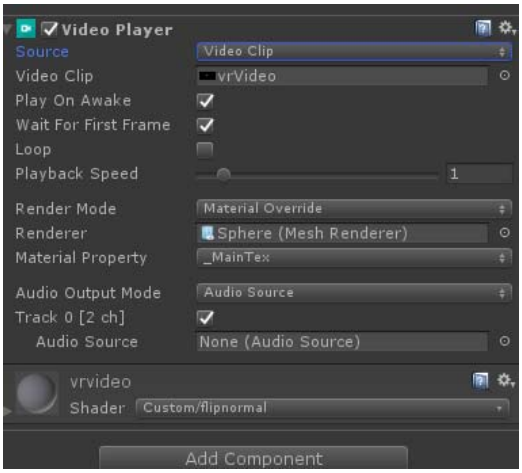

图 2.39 Video Player 组件

将 GoogleVR SDK 导入，并更改一些相关的设置：依次选择 Unity 菜单栏中的 File-Build Settings，将当前的场景添加进列表中，选择 Android 作为输出构建平台。在将平台切换完成之后，点击 Player Setting 打开播放器设置，将 Other Settings 下的 Virtual Reality Supported 勾选，并点击下面的加号，选中 Cardboard 添加至列表。

在将上述设置都完成后，将 GoogleVR/Prefabs 文件夹下的 GvrViewerMain 预制件拖拽到场景中，在 inspector 中将坐标设置为球体中心(0,0,0)。最后导出 APK 到 Android 设备上就可以在手机端观看视频了。GoogleVR 的其他功能用户也可以动手实现，并且现阶段的许多大厂商都已经在研发相关 VR 全景视频播放的内容，许多 unity 的 SDK 可以用来实现全景视频的播放并且开发者使用起来也十分方便。

Unreal Engine 4

UE(Unreal Engine)是全球顶尖游戏引擎，占用全球商用游戏引擎 80%的市场份额。虚拟引擎是美国 Epic 游戏公司研发的一款高标准游戏引擎，渲染效果强大，采用了 pbr 物理材质系统，不仅能够用于制作主机游戏，PC 游戏，手机游戏，还能够涉及高精度模拟，可视化与设计表现，无人机巡航等诸多领域。并且 UE4 为手柄、VR 控制器提供了良好支持。下面介绍如何在 UE4 中播放全景视频。

首先在 Content Browser 中展开 Sources Panel，并且在 Content 下创建名为 Movies 的文件夹，如下图：

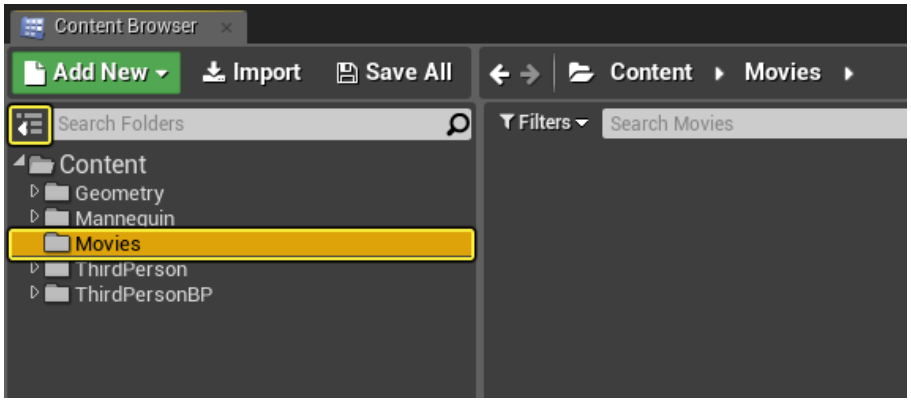

图 2.40 Content Browser

在 Movies 文件夹上点击鼠标右键，在出现的菜单中选择 Show in Explore。接下来将全景视频文件或者其他格式的视频文件拖到 Content/Movies 文件夹下，好能够保证视频能够被正确打包。

在 UE4 的项目工程中，在 Movies 文件夹上右键鼠标点击，在 Media 下选择 File Movie Source：

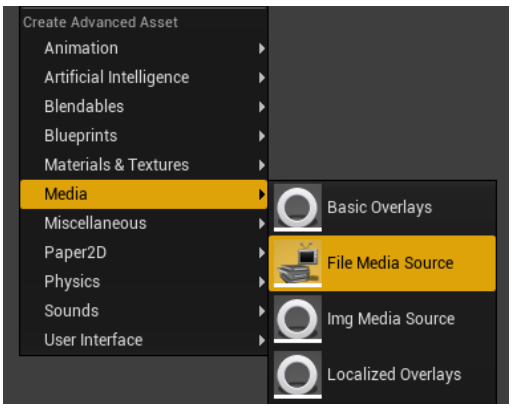

图 2.41 选择文件来源

将这个资源命名为 SampleVideo，打开以后，在 File Path 处将其定位到 Content/Movies 文件夹中的视频：

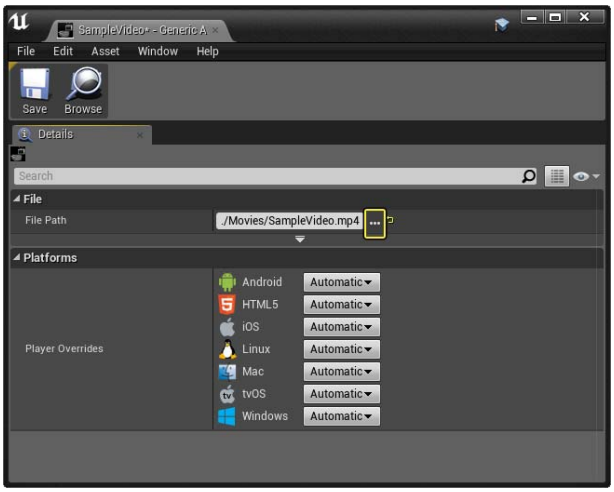

图 2.42 定位文件

接下来在 Content Browser 出点击鼠标右键，然后在 Media 下选择 Media Player 资源：

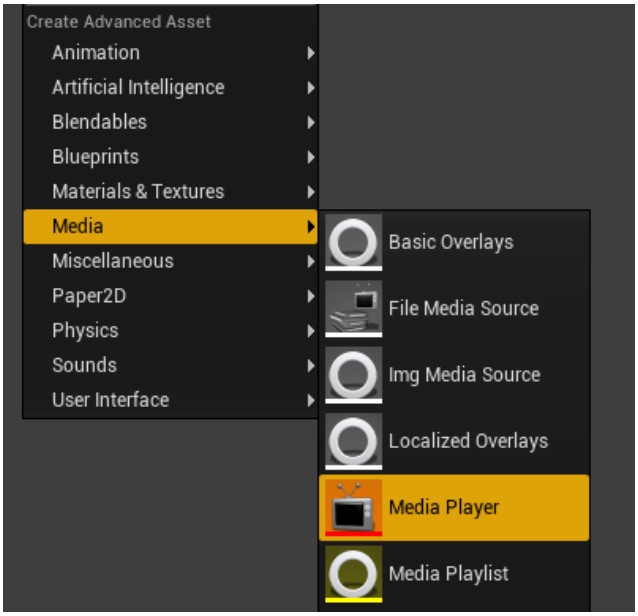

图 2.43 Media Player

在弹出的 Create Media Player 窗口中，点击选择 Audio output SoundWave asset 和 Video output Media Texture asset。通过选择这两项，会自动创建 SoundWave 和 MediaTexture 资源，并且会与播放视频所需的 MediaPlayer 资源相关联。

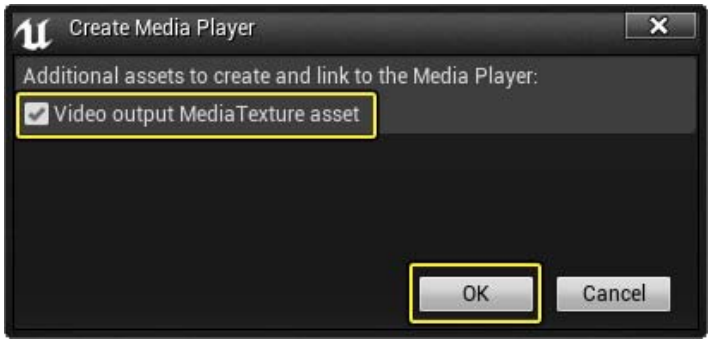

图 2.44 关联 asset

接下来对新的 MediaPlayer 资源命名，这里将其命名为 MyPlayer，相对应的 SoundWave

和 MediaTexture 也会改变；打开 Media Player 资源，双击 Media Source 资源，然后视频就会播放，并且在右下角的 Details 面板里，Output 部分 SoundWave 和 Video Texture 会被自动赋值。

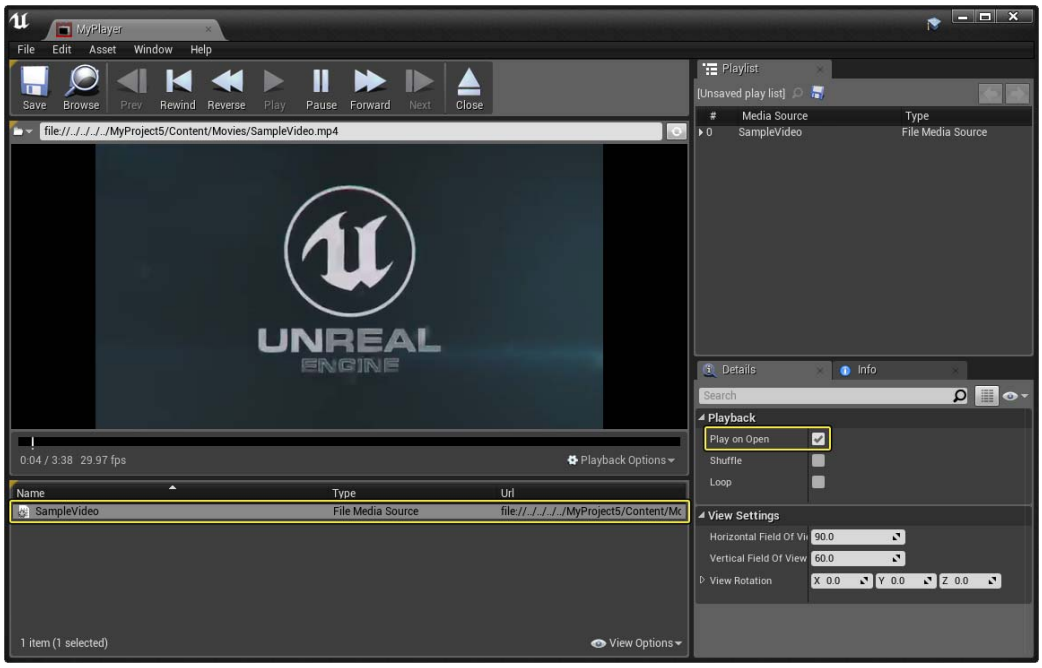

图 2.45 命名资源

按住 Ctrl 然后同时选择 SoundWave 和 Media Texture 资源,将其拖放到创建的球体 Mesh 上(自己提前创建一个球体即可)，这将自动创建 Material 并将其赋予到 Static Mesh 上。接下来再点击工具栏中的 Blueprints 按钮然后点击 Open Level Blueprint：

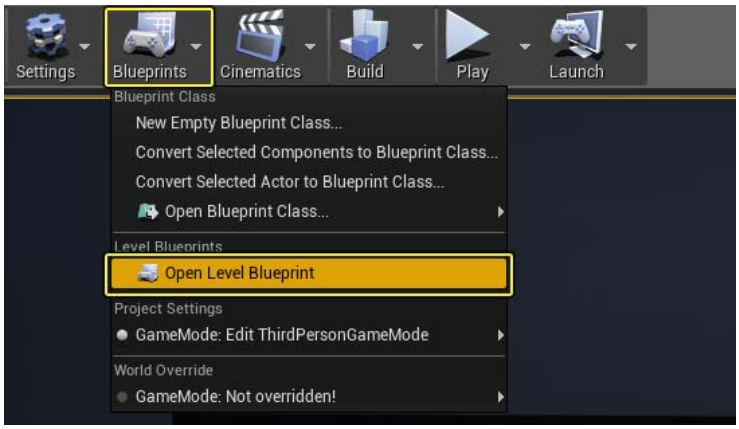

图 2.46 Open Level Blueprint

添加一个 Media Player Reference 类型的 Variable，并且将其命名为 MediaPlayer，并且将其设置为 MyPlayer 的 MedaiPlayer 资源。然后需要 Compile 这个资源，才能够设置 Default Value。

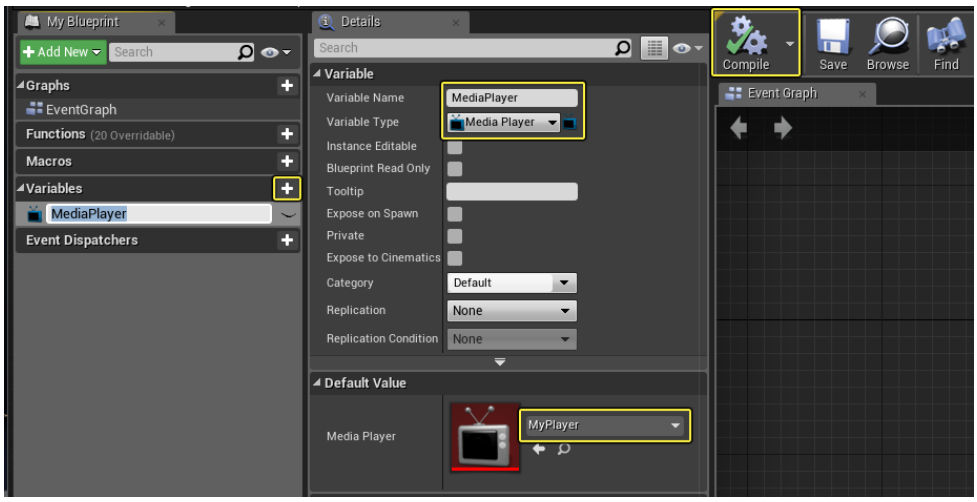

图 2.47 编译资源

然后按住 Ctrl 并将变量 MediaPlayer 拖放到 Event Graph 窗口，然后点击鼠标右键并添加一个 Event Begin Play 节点：

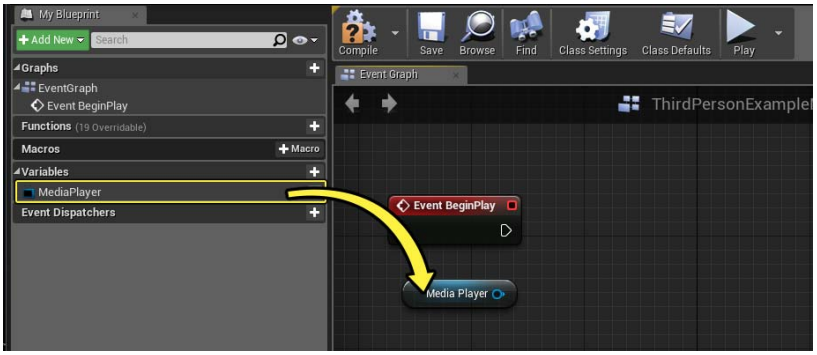

图 2.48 添加节点

拖动变量 MediaPlayer，然后使用 OpenSource 节点，将其 MediaSource 设置为 SampleVideo，如下图所示：

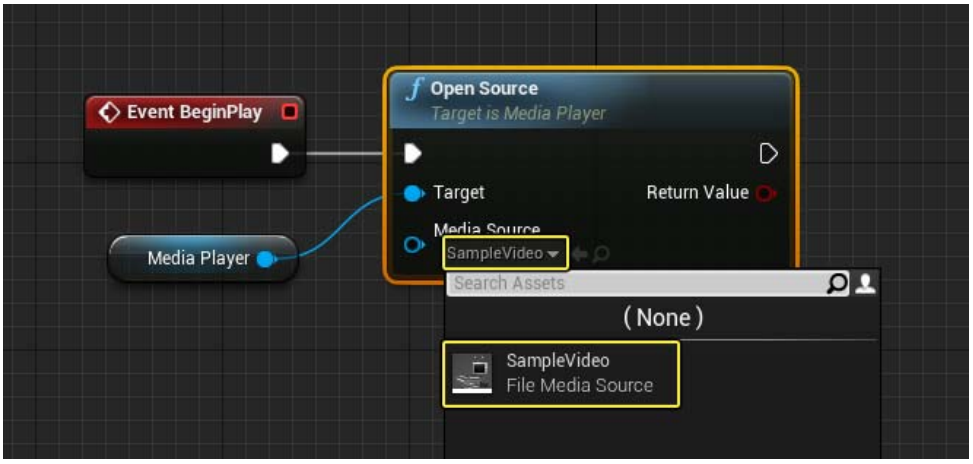

图 2.49 设置 Sample Video

最后，关闭 Level Blueprint，然后在编辑器内点击 Play 按钮即可。

根据以上介绍的过程来看，使用 UE4 来播放全景视频原理上是和 Unity 一样的，但是实际操作起来的难度却有所增加，而且相对 Unity 来说，UE4 的成本更高，使用难度更大，需要更多的时间去熟练相关的操作。

WebVR

WebVR 顾名思义，就是 web+VR 的制作方式，使用户能够在网页上观看全景视频或者全景图片。WebVR 有两种体验方式，一种是 VR 模式，另一种是裸眼模式。VR 模式下，可以在移动设备上观看，比如 Cardboard 等来体验手机浏览器的 WebVR 网页，而浏览器会根据陀螺仪的参数等来获取用户的头部的倾斜角度和转动的方向进一步告知页面需要渲染哪一个朝向的场景；还可以在 PC 端观看，比如通过佩戴 Oculus Rift 的分离式头显浏览连接在 PC 主机端的网页，现在 WebVR 还没有广泛应用，支持 WebVR API 的浏览器主要是火狐的 Firefox Nightly 和设置了 VR enabled 的谷歌 Chrome beta。还有就是裸眼模式，在 PC 端，裸眼模式应该允许用户可以使用鼠标拖拽场景，实现视角的转动；在移动端，则应让用户使用 touchmove 或旋转倾斜手机的方式来改变场景视角。

需要使用到的框架有 Three.js，它是构建 3D 场景的框架，封装了 WebGL 函数，简化了创建场景的代码成本，需要引入的插件有 three.min.js 和 webvr-polyfill.js，后者提供了大量 VR 相关的 API，比如 Navigator.getVRDisplay()获取头显信息的方法。现在已经有很多能够支持全景视频播放的网站，比如有 YouTube，steam 等平台，国内的优酷，可以定制的 play2VR 等等，这些都可以很好地播放全景视频。

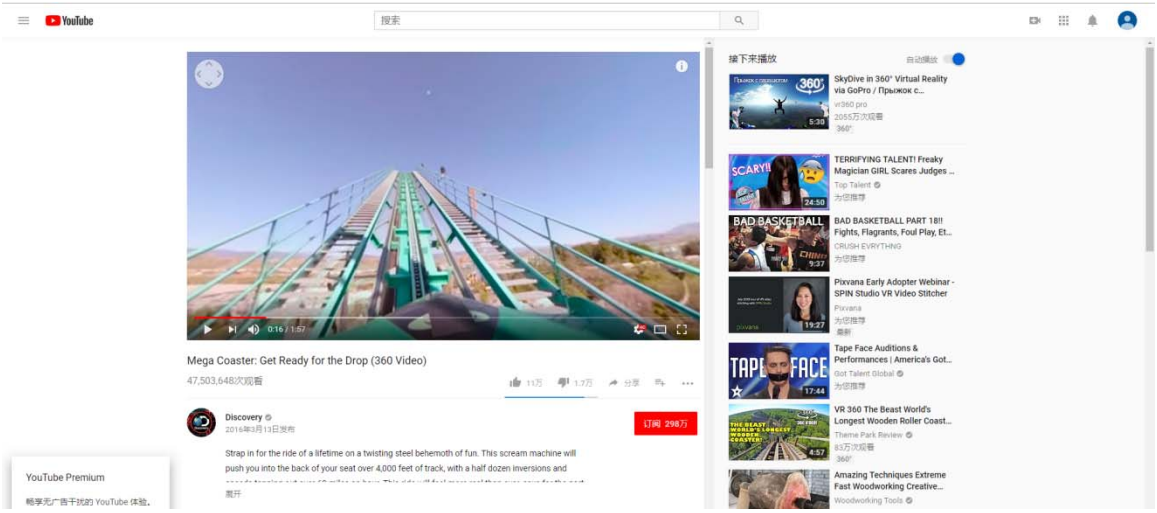

图 2.50 YouTube 中的全景视频

就像 YouTube 网站一样，当把鼠标移动到视频上面时，会出现能够拖动视角的手性标志，进而点击鼠标左键拖动即可。

现阶段各大厂商均已加入到了全景视频的发展之中。在可预见的未来，VR 将会发展迅猛，是未来科技发展的重头戏，全景视频的呈现也将会变得越来越简单和普及，人们将会更轻易地享受到观看全景视频的乐趣。

2.5 音频相关技术

2.5.1 声场

传统的音频信号一般使用麦克风进行捕获和记录，而最终通过扬声器等设备回放声音信息。这是目前仍比较常用的一种做法，但是这种方法的前提是声音对象可以通过点源捕获，然后由一个（或少量）点源渲染便可充分表示。

声场表示则消除了这种假设，允许捕获和回放完整的音频声场，用户就像在现实世界中体验一样。

2.5.2 声场通信场景

以下是涉及声场捕获和重建的多种场景：

1.声场捕获和记录：以尽可能高的保真度捕获和记录声场。没有带宽限制且没有传输信道，但可能需要压缩来缓和数据量。传感器的数量将影响相应情况下的要求。

2.声场重放和重建：这是捕获和记录的逆过程，这种情况下声场是通过不受带宽限制的媒体播放的。

3.声场单向广播/流传输：这是一对多场景，该情况下声场通过带宽限制的信道传输，最终传至多位置的终端。

4.声场双向通信：这种情况设想了双向通信的情形，其中任意一端都具备发送和接收声场的功能。

2.5.3 声场传感器配置

考虑上述所有情况的两个关键的方面：声场捕获和声场重建。

“平面”上的传感器

声场捕获和重建的简化解释是想象听众和感兴趣的声音对象之间的无限平面，当声波从物体传播到听者时，它们穿过平面。如下图所示：

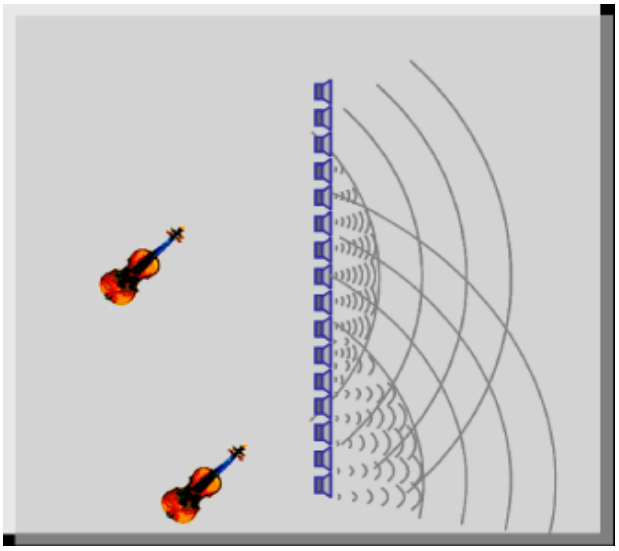

图 2.51 穿过平面的波场

传感器在平面中的最有效放置是六边形填充，如下面的图 2.52 所示，其中“X”表示传感器位置。

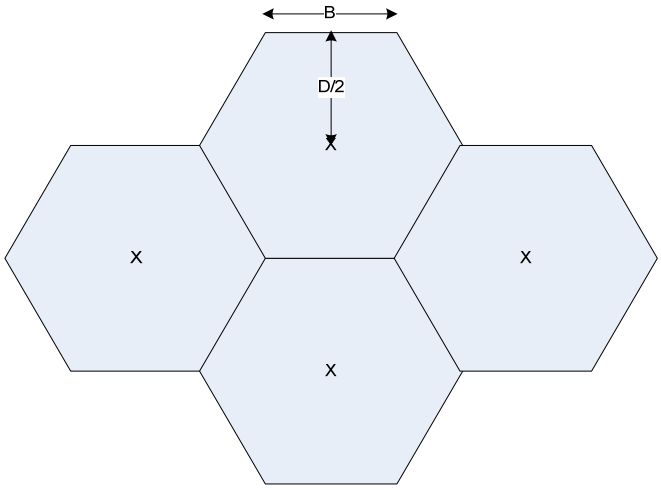

图 2.52 具有六边形拼接的传感器

虽然人类认知的真实世界是模拟的，但研究人员总希望将这个世界各个方面的信息表示为一些数字序列。对于单个音频传感器，我们可以假设 48kHz 的（时间）采样频率 fs 足以表示音频信号，因为人们普遍认为人类听觉的频率范围是 20Hz 到 20kHz。

声场的捕获同时需要最小的时间采样频率和最小的空间采样频率。空间采样频率由传感器的空间位置确定。假设传感器使用六边形平铺均匀分布在平面上，如上图 2.52 所示。任意一个传感器到其最近传感器的距离 D 为：

$$B^2 = (D/2)^2 + (B/2)^2 \tag{2.1}$$

或

$$D = B\sqrt{3} \tag{2.2}$$

其中 B 是六边形边缘的长度， $D/2$ 是从六边形中心到边缘中点的距离。

一个更简单的分布是仅考虑水平线而不是平面上的传感器。在这种情况下，传感器将以距离 D 分开放置。

在平面或线性配置中，如果传感器被间隔得太远，则重建的声场将遭受“空间混叠”，因为它不能精确地捕获波长比临界距离 fc 更短（频率更高）的信号分量。这种情况在下面的图 2.53 中得以说明。该图显示出了以角度 θ 照射在传感器平面上的平面波。波长是 λ，波峰逐个测量。

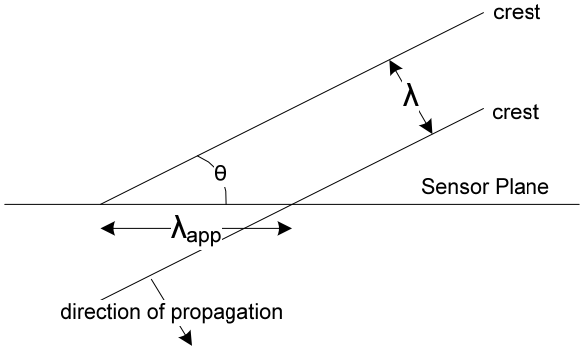

图 2.53 以角度 θ 撞击传感器平面的平面波

由于入射角的存在，平面上的传感器感应到更长的波长 λ_{app}，即：

$$\lambda_{app} = \frac{\lambda}{\sin\theta} \tag{2.3}$$

入射角决定了临界频率：

$$f_c = \frac{c}{2D\sin\theta} \tag{2.4}$$

或

$$D = \frac{c}{2f_c\sin\theta} \tag{2.5}$$

其中 c 是声速，D 是传感器间距。临界频率 fc 是传感器可以在没有混叠失真的情况下检测到的最大频率。随着入射角 θ 增加，当 θ = 90 度时，临界频率降低到极限：

$$f_c = \frac{c}{2D} \tag{2.6}$$

虽然我们假设了一个无限的传感器平面（或线），但这在实际中是不可能的。有限范围的传感器线阵或平面在声场捕获和重建中会引起“截断效应”失真。

在信号处理术语中，这是空间域中的频谱泄漏，并且是由于以矩形函数作为窗口函数应所引起的。如果外部传感器或转换器的检测或再现水平降低，则可以减少泄漏程度。这对应于信号处理中使用边缘逐渐变细的窗口函数。

球面上的传感器

平面传感器的一种变体是球面传感器。当球体的半径 R 变为无穷大时，球面就变成一个平面，因此有很多共同之处。然而，球面传感器的概念存在更吸引人的方面，这里说明两种不同的场景。

第一种情况是以用户为中心，捕获影响虚拟用户的声场，然后在不同的时间和地点为实际用户再现声场。在这种情况下，声场由距离虚拟用户 R 的物体产生。

第二种情况是以发声物体为中心，捕获从实际声音对象发出的声场，然后在不同的时间和地点再现以模拟虚拟对象。在这种情况下，声场由实际声音物体产生并且被用户感知到一系列远离自身虚拟对象的发声物体。

在每种情况下，我们可以认为声场是由半径为 R 的球体表面上的传感器捕获的。在以用户为中心的情况下，传感器（即麦克风）面向外，或者其方向性图案朝外，以捕获球体之外的声音。在以物体为中心的情况下，传感器面向内，或者其方向性图案面向内，以捕获球体内部的声音。如果捕获传感器具有全指向性，则两种情况下的传感器配置是相同的。

对于声场重建，以用户为中心的情况，球体表面上具有转换器（即扬声器），其具有面向内的方向性图案，而以物体为中心的壳体将具有面向外的方向性的转换器。

下面的图 2.54 显示了以用户为中心或以物体为中心的情况的传感器/传感器配置。

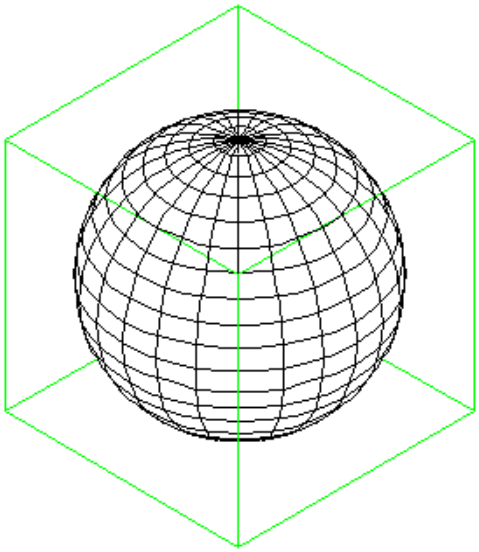

图 2.54 封闭球体上的传感器。声源对象位于球体的中心，用户位于外部，或者用户位于球体的中心，源对象位于外部。

如果值 R 足够大，则可以为单个，少量甚至大量个体定制以用户为中心的场景。如果 R 足够大，则可以为单个物体或多个物体定制以物体为中心的情况。或者，为了达到声场捕获的目的，每个发声物体可以拥有一个球体，即每个发声对象可以有类似的单个球体用于重建。

如果半径为 R 的球体的表面均匀地铺设有六边形，那么传感器的数量 N 是球体的表面除以六边形的面积。使用图 2.52 中的尺寸 D 和 B，以及上面的临界频率 fc 的等式，则 N 为：

$$N = \frac{4\pi r R^2}{(B^2)(3\sqrt{3}/2)} = \frac{4\pi R^2}{D^2(\sqrt{3}/2)} = (\frac{2}{\sqrt{3}})\frac{4\pi R^2}{(\frac{c}{2f_c \sin\theta})^2} \tag{2.7}$$

可以看出，关于最大入射角的假设对于确定所需传感器的数量是至关重要的。但是，如果需要高达 60° 的入射角，那么半径为 4 米的球体将需要 N = 600 000 个传感器才能完全捕获具有 20 至 20kHz 带宽的声场。

2.5.4 沉浸式媒体中的音频

我们所处的现实环境中声音来自四面八方，因此对于周围的环境状况和发生的事能够产生直接、准确的判断，在沉浸式的虚拟环境中，同样需要让用户听到来自四面八方的声音，才有助于在虚拟环境中产生真正的沉浸感。

关于如何在 VR 中实现这一点，首先来看我们在日常的 3D 电影、3D 游戏中已经接触到的 3D 音效。看 3D 电影的时候，由于声源有确定的空间位置，声音有确定的方向来源，因此人们能够辨别到声源的方位。5.1 环绕声的效果比双声道立体声更加“3D”，7.1 环绕声比 5.1 环绕声更加“3D”，在一定上限范围内，音箱数量越多，3D 环绕声系统的效果就越好。但这样的环绕声系统的音箱位置是在同一平面上，因此现在又有了“杜比全景声”，在影厅的天花板上也装有音箱，这样观众就能听到来自头顶的声音，与环绕声是一样的设计思路。

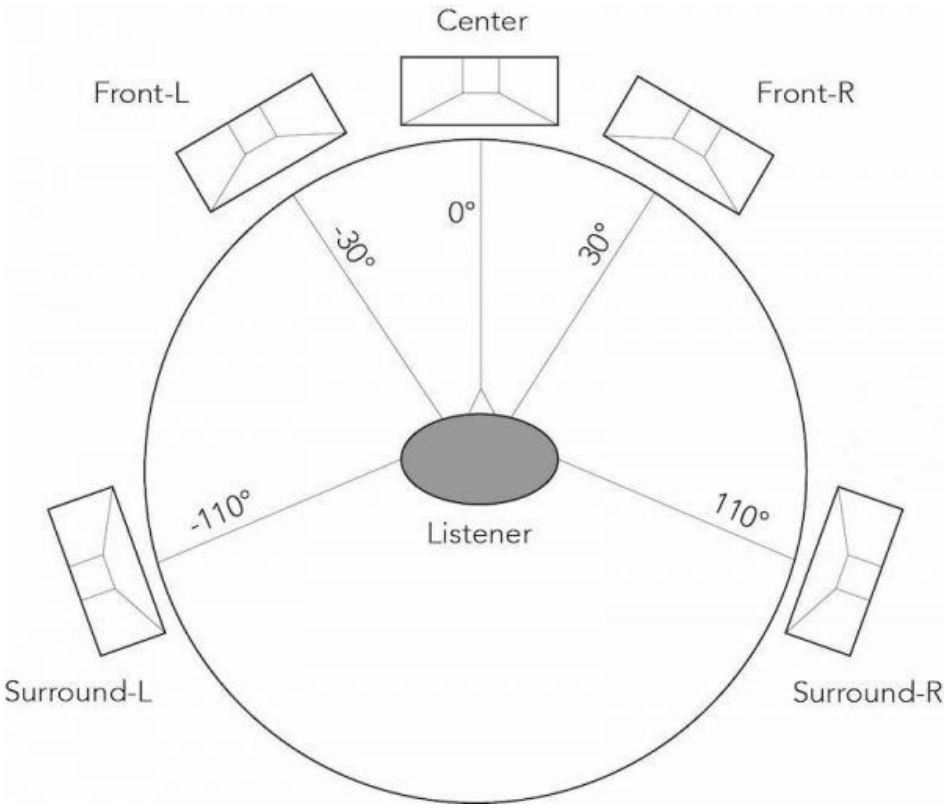

图 2.55 5.1 环绕声

而在 VR 中，观众处于场景中心，可以自主选择观看的方向和角度，用户要通过头显加耳机的方式感受 VR 体验，就需要在双声道立体声输出的耳机上听到来自各个方向的声音。

另一方面用户需要来回转动头部或者有大幅度的身体运动，因此还要考虑身体结构对于声音的影响。因此在 VR 中需要解决关键的两个问题，一个是怎么放，一个是怎么听。

首先，声音怎么播放的问题。在 VR 中制作声音时，要以用户为中心，在整个球形的区域内安排声音位置，确定某一方向基准后，画面内容与用户位置也就是相对确定的，以此来定位，既有水平方向的环绕声，也有垂直方向上的声音。通过水平转动和垂直转动这两个参数，就能控制视角在 360 度球形范围的朝向，以及与画面配合的声音的变化。

另一方面，用户只有一副耳机，要实现电影院里杜比全景声的效果，需要用到一项技术叫做 HRTF（Head-related Transfer Function“头部传送函数”），该技术能够计算并模拟出声音从某一方向传来以及移动变化时的效果，类似于一个滤波器，对原始声音进行频段上的调整，使其接近人耳接收到的听感效果，并通过耳机来回放。

基于这样的原理，不少厂商已经进行了尝试来创造 VR 中的音效。

Oculus

早在 2014 年，Oculus 授权 VisiSonic 的 RealSpace 3D 音频技术，并将其融入 Oculus Audio SDK 中。通过跟踪器上所发来的空间信息来处理声音信息，让听者觉得该声音是从这个物体中发出来的。这项技术非常依赖定制的 HRTF，通过耳机来再现精准的空间定位。

NVIDIA

到 2016 年 5 月，NVIDIA 就推出了一个专门用于虚拟现实场景，第一个基于物理技术的声学仿真技术“VRWorks Audio”，借鉴了光线追踪渲染的思路，充分考虑了 3D 场景的渲染，通过将音频交互映射到 3D 场景中的物体上，使音频听起来更加自然。用户不断移动，能够

听到回声的变化以及带来的空间感，除了能够判断声音是由该物体发出之外，还能判断出物体的方向、远近等等跟多的信息。

AMD

与英伟达类似，2016 年 8 月，AMD 在发布了一项名为 TrueAudio Next 的实时动态声音渲染技术，让虚拟现实中的声音和画面更为同步。

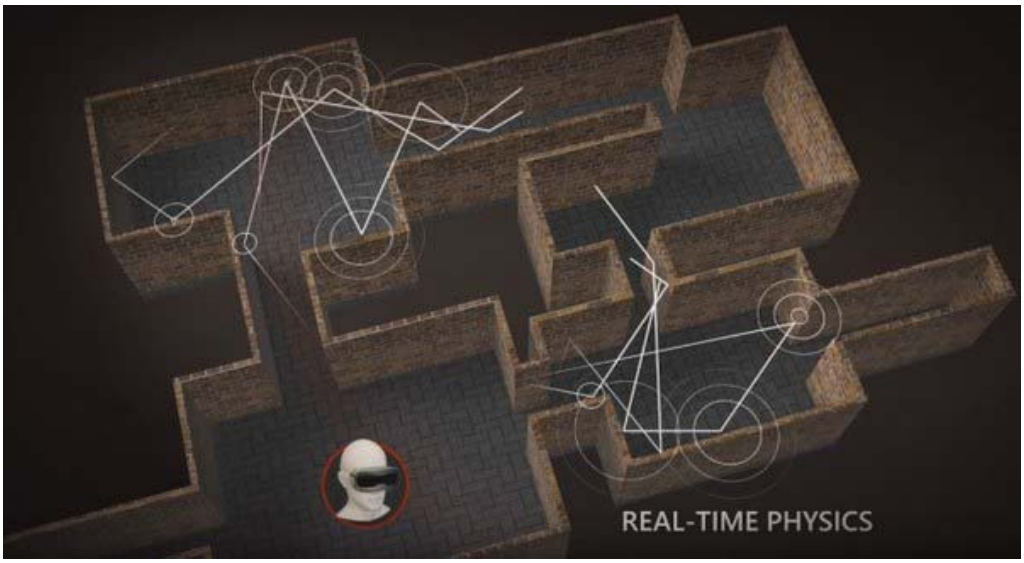

图 2.56 实时动态声音渲染技术

该技术同样使用物理方式模拟，让渲染的声音无限接近真实环境的声音，在虚拟建模中进行多次反射，利用 Radeon Rays 光线追踪技术让系统辨别 VR 空间布局并定位空间中的物体。AMD 已将该技术开源。

谷歌

近年谷歌也与音频公司 Firelight 和 Audiokinetic 合作，推出一个 VR 音频插件。开发者利用该插件可以根据虚拟空间大小、材料以及对象位置的改变来调整声音，营造更加逼真的氛围。该插件可以无缝集成到 Unity 和 Unreal 引擎中，使用时开发者只需要对 3D 音频进行简单调节，能够很轻易地创造空间音频。

此外，谷歌公布了面向 Web 端的 Omnitone，一个跨浏览器支持的开源空间音频渲染器。同样使用 HRTF，但是他们主要解决的问题是，在已有的浏览器里引进环绕立体声技术，同时不能干扰浏览器原本的运行。

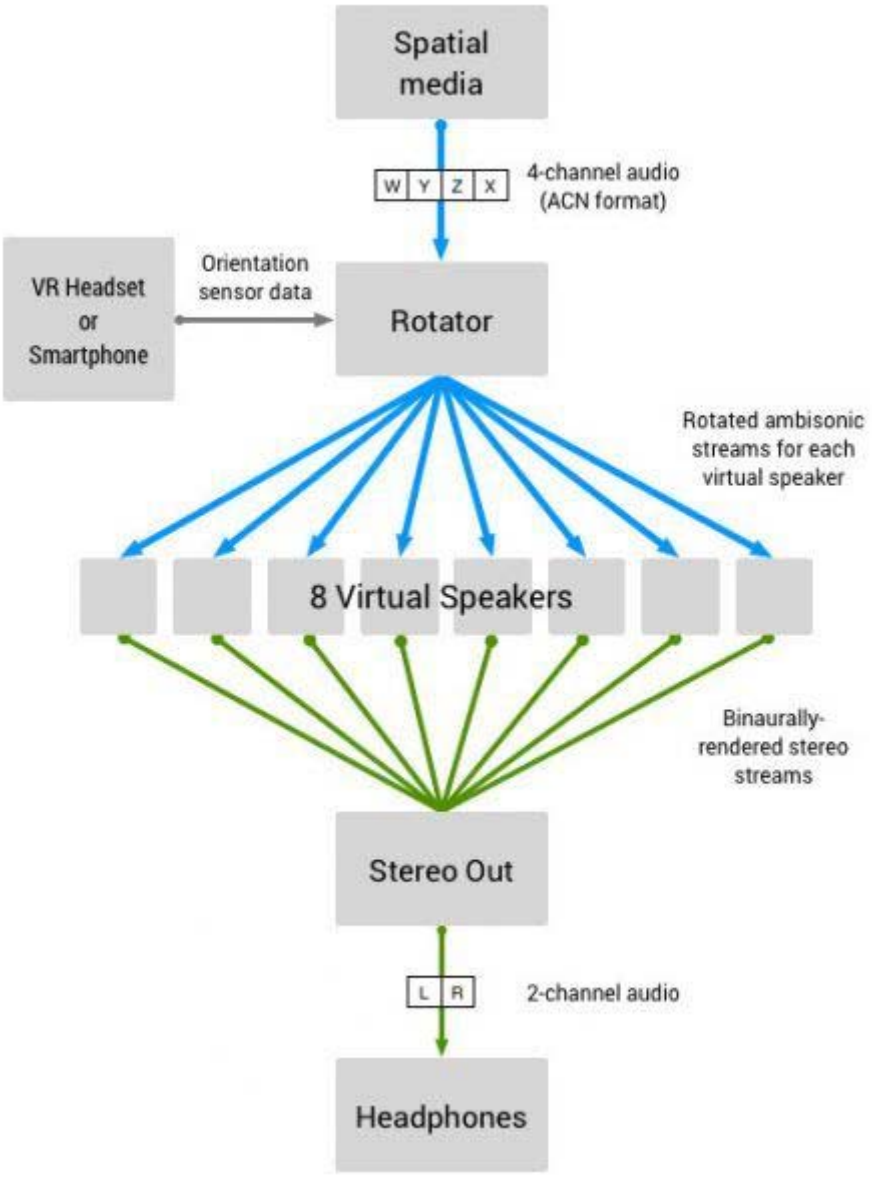

图 2.57 谷歌 Omnitone

上图是谷歌 Omnitone 的解决方案。在环绕立体声里包含了 4 种声道，可在任一扬声器中解码。谷歌在 Omnitone 中设置了 8 个虚拟扬声器来渲染双耳音频流，将 VR 头显中的方向传感器数据与解码器无缝衔接，完成音场转换，从而让用户通过耳机就能体验到空间感。

Valve

Valve 此前曾收购了音效公司 Impulsonic，Impulsonic 有一个基于物理的声音传播和 3D 音频解决方案，名为“Phonon”，近日，Valve 开放了 Photon 音效工具的后续产物 Steam Audio SDK。该方案能够通过空间音效增强 VR 沉浸体验，允许游戏的音频与场景几何体建立交互与反弹回音，从而增强体验。Steam Audio 支持 Windows、Linux、macOS 和安卓等多个平台，也不局限于特定的 VR 设备和 Steam。

小结

目前已有的音频技术可以实现 360° 全景声，可以通过声音辨别方向、距离。但是 VR 音频技术要求不仅仅能够在提供 VR 环境中物体的位置信息，更要反馈出更多的空间环境状

态。

以 VR 游戏为例，当光线越来越暗，视觉必定受到限制，这个时候就要靠音频来确定环境状态，脚步声、风声、动物的叫声等都能为玩家提供信息，诱导下一步的行动和交互。因此，精准有效的音频技术在 VR 中特别重要，不仅仅是游戏、视频，还有其他例如教育、社交等领域，VR 音频技术也需要进一步的成熟。

2.6 数据转换技术

2.6.1 SLAM

即时定位与地图构建（Simultaneous Localization and Mapping，SLAM）是根据多点云构建 3D 地图（点云）的过程，每个点都由不同的定位（扫描仪位置）获取，但不必精确地知道这些定位。 SLAM 的作用是融合这些点云，逐渐更新预设的定位，直到不同点云之间的距离最小化。图 2.58（左）显示了同一场景中两个未对准的点云，图 2.58（右）显示了 SLAM 对齐的效果，这有效地增加了所得点云的密度。

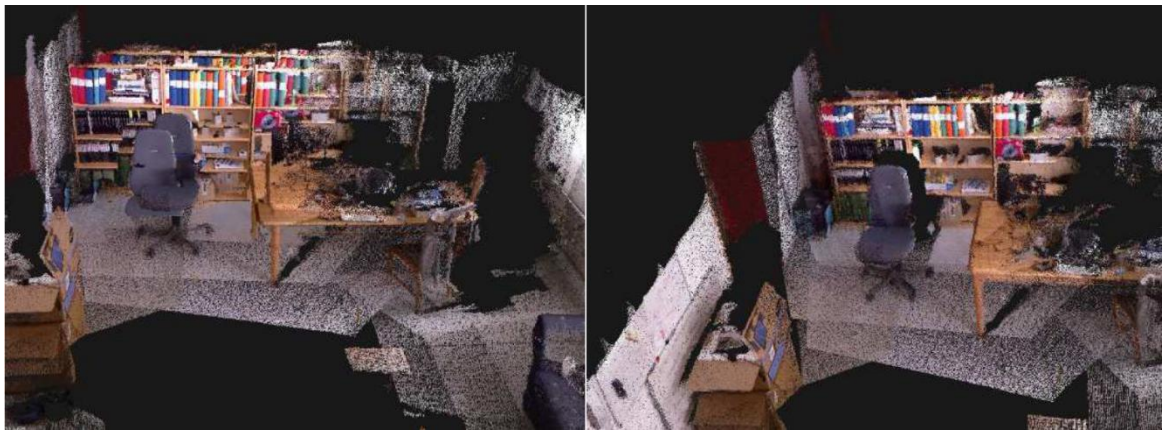

图 2.58 对齐前（左）和对齐后（右）的点云的 SLAM 融合

SLAM 在激光雷达扫描中非常有用，其中点云的密度随着扫描物体与扫描仪定位之间的距离而减小。为了获得更均匀的点云，可以从不同位置扫描场景，并使用 SLAM 将获取的点云融合在一起。然后可以对得到的高密度点云进行编码等操作。

2.6.2 点云到深度图

如图 2.59 中所示（右上），由激光雷达扫描仪捕获的点实际上是以球坐标（即角度和深度）获取的。大多数情况下，这些坐标在扫描仪内部转换为笛卡尔坐标(x,y,z)，参见图 2.59（左下）。

然而，笛卡尔坐标仅是中间坐标（可能在 SLAM 预处理之后），最终仍会被转换回投影深度图，表示每个点到平面的距离。图 2.59（右下）显示了这种深度图，以及进一步用于编码的低通和高通二次采样版本（小波分解）。

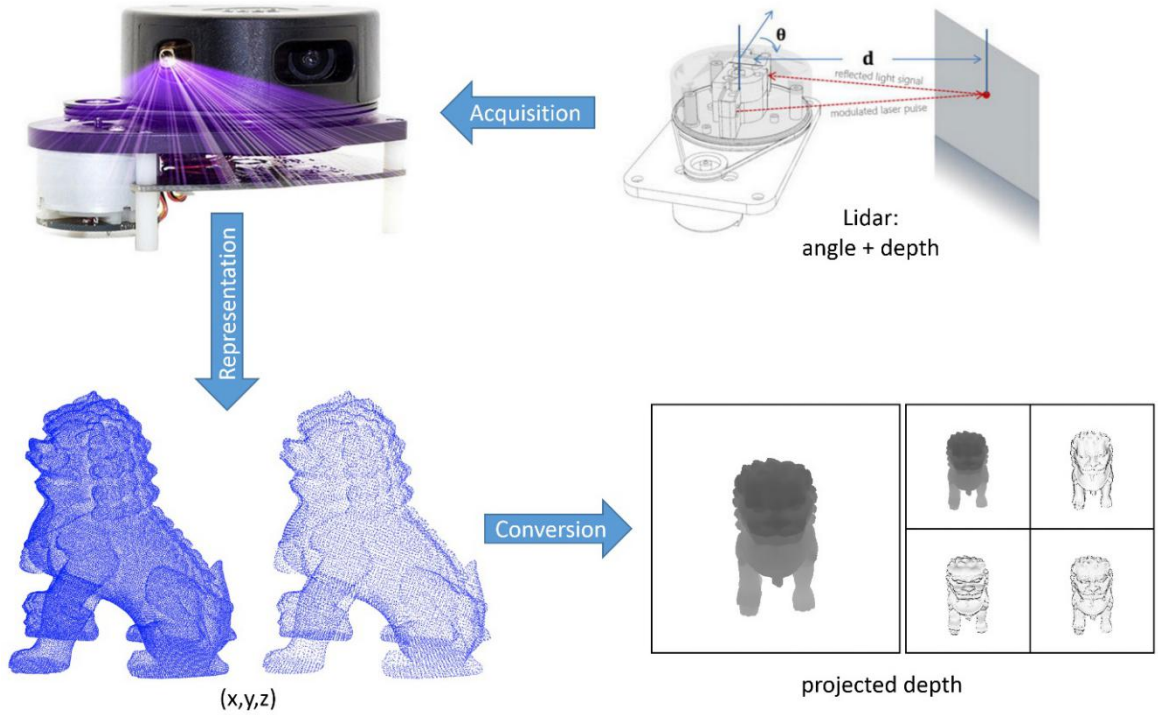

图 2.59 激光雷达获取（角度+深度）点云通常用（x，y，z）表示，但也可能转换为投影深度图。

2.6.3 点云到三角形网格

图 2.60 显示了将点云转换为三角网格的效果，这样便可以在经典的 OpenGL 处理通道中轻松渲染。要注意的是，良好重建封闭区域是较为困难的，这一点可以通过上述的 SLAM 技术来处理。

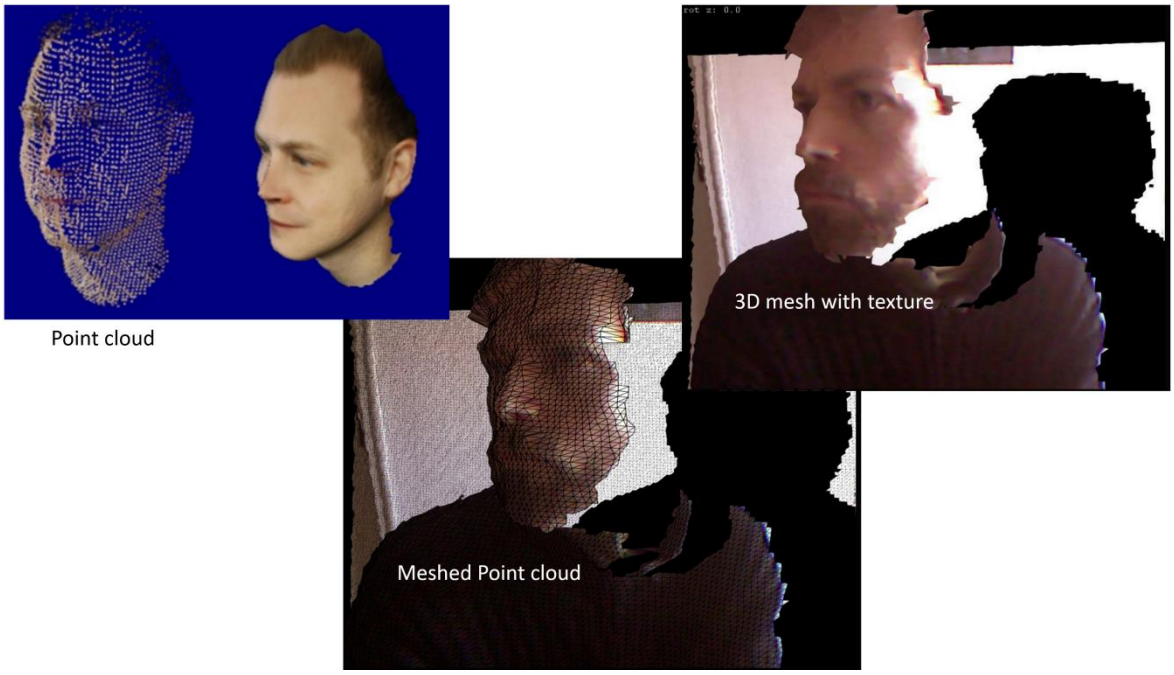

图 2.60 将采集的点云转换为 3D 网格

2.6.4 点云到平面基元

在某些应用中，例如建筑信息模型（BIM），提取表示对象的平面基元比保持点云的各个点更有益，如图 2.61 所示。进一步而言，将多个点归并至同一个平面基元的做法也可以在基于网格的编码中产生更好的性能。

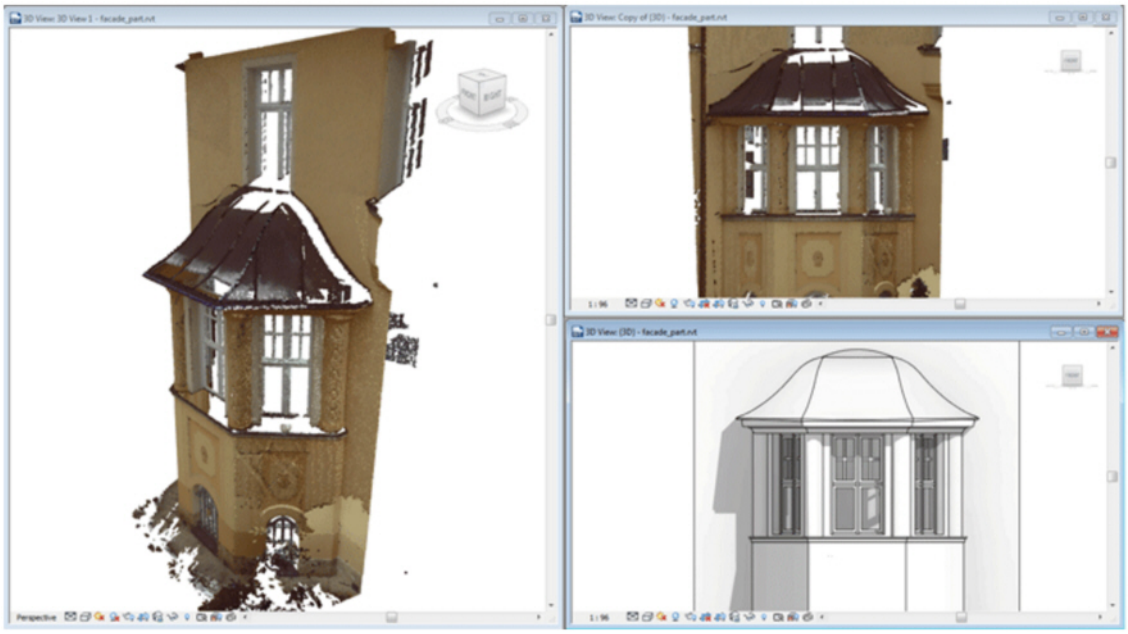

图 2.61 点云（左）的平面基元提取（右下）

为了编码上的目的，图 2.62 的背景与前景对象被分离开，并将背景拼接成一个可用作全景图像的纹理图，这在 VR 中经常进行，只有这样大尺寸图像的窗口区域才能在 HMD 设备上可视化，参见图 2.63。图 2.62 的其余点则可以被视为常规点云进行后续操作。

图 2.62 根据深度信息图（a）和投影视图（b）的混合点云和分段平面表示（c）

图 2.63 虚拟现实的全景纹理图：在任何时间戳，纹理图的一个区域在头戴式设备中呈现

2.6.5 点云与光场

通常，点云中的点不需要在所有方向上具有均匀的颜色。实际上，在自然界中，许多现象是镜面的，在所有方向上都具有不均匀的颜色分布。这样的信息可以通过所谓的 BRDF 函数来表示，以告知点云中的每个点在每个方向上的颜色传播。从某种意义上说，这个点本身就会发出一个“光场”，所有点拥有的场的结合会产生一个所谓的光场。这清楚地表明了点云和光场之间的双射等价关系。

如图 2.64 所示，点云/光场表示空间中的每个立体像素 (x,y,z)，其颜色 C 在各种光线方向 (θ,ϕ) 下会有所变化，即颜色 C 是 5 参数的函数：

$$C = f(x,y,z,\theta,\phi) \tag{2.8}$$

然而在实际中，该函数可以减少至 4 个参数，因为光线一般在整个传播过程中保持不变（除非存在微粒，如雾，这会降低传播路径上的光强度）。因此，每条光线可以通过其与相机平行的平面的交点 (s,t) 及其传播角度 (θ,ϕ) 来表示：

$$C = f(s,t,\theta,\phi) \tag{2.9}$$

光线也可以用两个平行平面的交点 (s,t) 和 (u,v) 表示，见图 2.64：

$$C = h(s,t,u,v) \tag{2.10}$$

后者是文献中最常使用的光场数据表示法。

总之，除了利用点 (x,y,z) 和方向 (θ,ϕ) 表示对应的颜色外，也可以用它与两个平行平面的交点 (s,t) 和 (u,v) 表示从这一点发出的光线。

此外，还可以在图 2.64 中观察到点云或光场表示的等效性，以及从摄像机视点的纹理图呈现的深度信息。

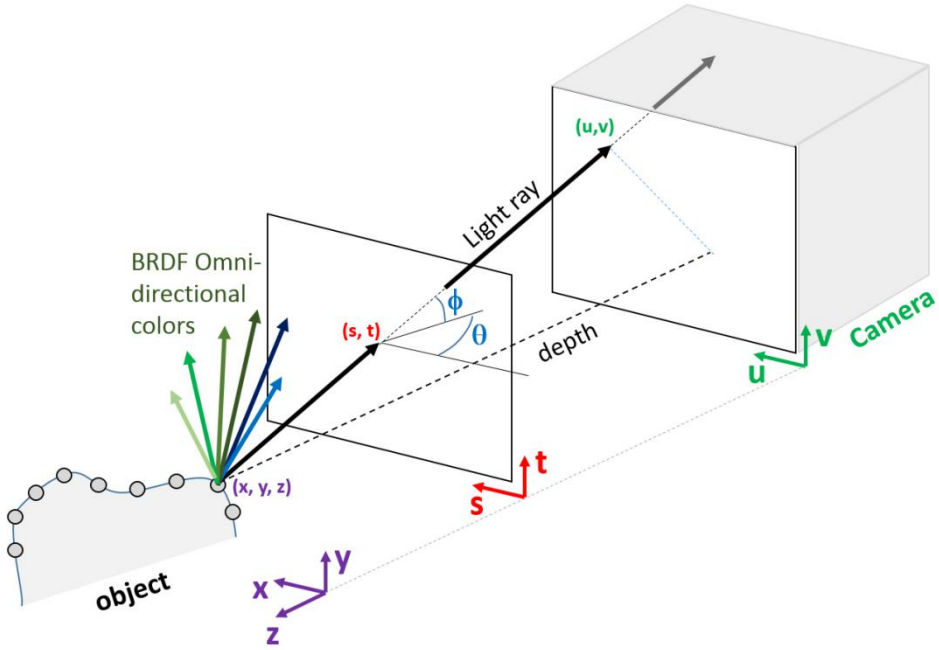

图 2.64 点云中的点发出的光线

2.6.6 光场到深度映射

如前一小节所述，深度信息可以从光场中恢复。这种深度信息通常用于基于深度图像的渲染（DIBR）。

考虑置于拍摄场景前的线性相机阵列的简单情况。将所有位置拍摄的图像堆叠在一起（沿着 u 轴）便创建了图 2.65 上部所示的图像堆栈。

这种图像堆栈的一个水平切片对应于具有许多对角线的所谓的核面图像（EPI），如图 2.65 底部所示。每条线表明了立体元素从一个摄像机镜头跳到下一个摄像机镜头时是如何移动的。远处（大深度）的立体元素几乎不会移动，在 EPI 中形成几乎垂直的线，而前景中的像素在不同角度的镜头中将表现出大的差异，因此产生大斜率的对角线。因此，EPI 中的对角线的斜率提供关于场景中各像素的深度信息，这种方法也被证明对于稀疏相机布置条件下的虚拟视图创建是有用的。

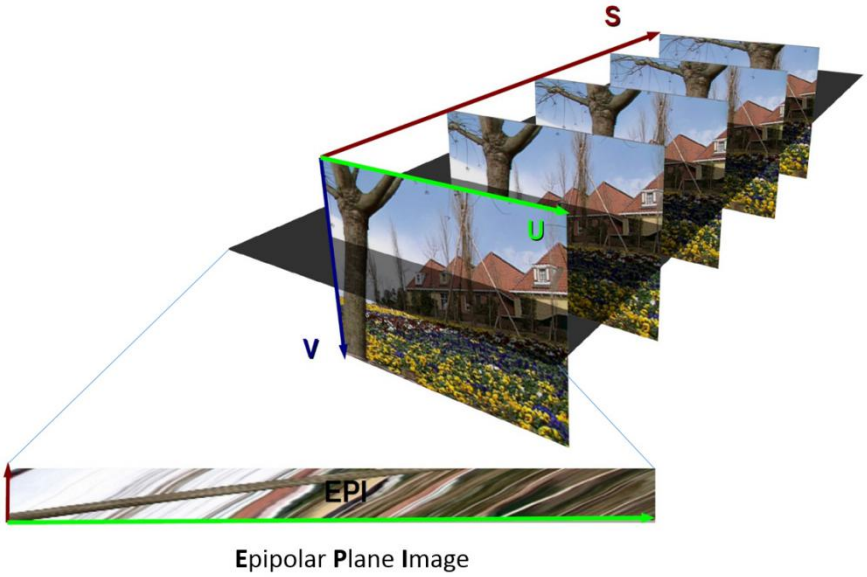

图 2.65 图像堆栈的 EPI 部分

然而，这种方法的一个难点是获得 EPI 的密集表示，如图 2.66（右），当仅具有有限数量的摄像机视图，即仅有 EPI 的二次采样版本时，如图 2.66（左），需要利用到特殊的线检测和插值技术。

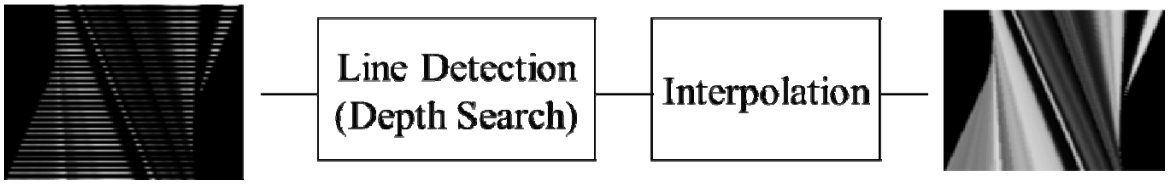

图 2.66 EPI 二次采样版本（左）的直线斜率检测（右）

点云用于沉浸式媒体的优势

1）时效性强：不需要耗费大量人力和时间来制作虚拟的 VR 场景，空间数据采集完毕，稍加处理即可投入使用。

2）真实性强，具备真实空间坐标信息，可对任意位置进行量测；信息采集于真实场地，可谓是对现实空间的完美复制。

3）时空意义，空间信息随着时间的变换而发生改变，点云能够存储不同时期同一空间位置的准确信息。

本章参考资料：

[1] Jhag, Pirogg (1 July 2011). "Gambling Trends for Online Casino". Retrieved 10 June 2018.

[2]https://en.wikipedia.org/wiki/Head-mounted_display

[3]https://blog.csdn.net/u014640129/article/details/23363661

[4]https://blog.csdn.net/liulong1567/article/details/50457730

[5]https://blog.csdn.net/liulong1567/article/details/50421643

[6]https://github.com/sjtu-medialab/VirtualReality-Chinese/tree/master/VirtualReality

[7]https://zhuanlan.zhihu.com/p/35679418

[8]https://zhuanlan.zhihu.com/p/27551369
[9]https://blog.csdn.net/cbbbc/article/details/70071240
[10]https://blog.csdn.net/nikoong/article/details/79776873
[11]https://blog.csdn.net/shenzi/article/details/5417488
[12]https://blog.csdn.net/dabenxiong666/article/details/55062609
[13]http://smus.com/vr-lens-distortion/
[14]http://www.sohu.com/a/41581937_105527
[15]http://vrguy.blogspot.com/2016/04/time-warp-explained.html
[16]https://blog.csdn.net/fr_han/article/details/50968110
[17]http://vga.zol.com.cn/577/5776982.html
[18]https://www.cnblogs.com/Anita9002/p/4975242.html
[19]http://www.tj108.cn/a/2017/1001/22196.html
[20]https://blog.csdn.net/caozhaodan/article/details/77647847
[21]https://docs.unrealengine.com/en-us/Engine/MediaFramework/HowTo/FileMediaSource
[22]http://blog.sina.com.cn/s/blog_142f5d2240102xqvu.html
[23]http://www.vrzy.com/vr/26418.html
[24]https://mpeg.chiariglione.org/standards/mpeg-i/technical-report-immersive-media