**第三部分**

**反射、折射和阴影**

任何光线追踪渲染都会导致关于效果是否支持以及如何支持的一些设计决策。光线追踪的主要历史诉求是它能够很好地处理阴影、反射和折射。然而，当你真的坐下来实现了一个支持这些效果的系统，你将面临一些不明显的设计决策。本书的这一部分描述了几个具体的达成这些设计决策的方法。

虽然对一个清晰的玻璃球进行光线追踪是一蹴而就的，但面对更复杂的模型，就有点令人头疼了。例如，一个简单的装有水的玻璃杯就呈现了光线在三种不同类型的材质之间传递的行为：水和空气，玻璃和空气以及玻璃和水。为了使折射正确，被追踪的光线不仅需要知道自己穿过了哪个接触面，还得知道是从哪种材质穿入哪种材质的。指望美术工作者用这些接触面建模是有问题的；想象一下在玻璃杯中装满水。第[11](#第11章)章《自动处理嵌套体积的材质》中描述了一个巧妙并且经过实战考验的方法来处理这个问题。

困扰几乎所有光线追踪程序的一个问题是当法线贴图产生了物理上不可能的表面法向量时应该怎么处理。显然，用硬编码的条件分支方案忽略这些不可能的法向量会导致不和谐的颜色不连续。每个光线追踪器都有自己的小技巧来处理这个问题。第[12](#第12章)章《用基于微表面的阴影函数解决凹凸不连续的问题》中，提出了一个基于简单的统计学方法的简洁实现。

光线追踪的屏幕空间方法使其在生成精确的屏幕空间阴影方面特别强大，而不会有阴影贴图方法带来的所有锯齿问题。然而光线追踪的计算能快到可以进行实时交互的程度吗？第[13](#第13章)章《光线追踪阴影：维持实时帧率》提供了关于如何做到这一点的详细解释。

大部分简单的光线追踪器都会从眼睛发出光线。通常这些程序几乎不可能产生焦散——就像我们用聚焦光图案制作装有液体的玻璃杯、游泳池或者湖泊。用“几乎”来形容是因为计算结果伴随了太多的噪点。但是，将光线从光源发射到环境中是一种产生焦散的可行方法。事实上，这甚至可以通过实时计算完成，正如第[14](#第14章)章所描述的《使用DXR在单一散射介质中实现光线引导的体积水焦散》。

总之，一个基础的光线追踪器是非常简单的。把光线追踪器部署到生产环境需要对基本效果进行一些小心处理，本书的第三部分提供了处理这些基本效果的几种有用的方法。

Peter Shirley

**117**

**第14章**

**以DXR在单散射介质中进行光线引导的体积水焦散**

Holger Gruen

NVIDIA

**摘要**

本章介绍了一种使用光线追踪和光栅化在单个参与散射的介质中渲染表面和体积焦散的混合算法。该算法利用基于DirectX的光线追踪（DXR），生成驱动硬件的曲面细分的数据，以自适应地改善为了切割体积焦散而渲染的三角形体积光束。在更进一步的渲染管线中，光线追踪还用于生成存储了被水表面反射或折射的光线的光线/场景交点位置的二级焦散图。

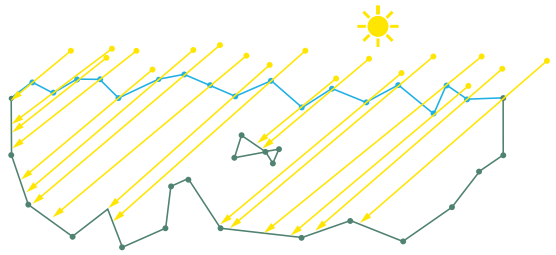
**14.1 介绍**

本章研究如何使用DirectX 12实时光线追踪API，DXR，简化当前用于在单散射介质中渲染实时体积水焦散的方法。体积焦散在过去已经被广泛地研究[2,5,6,10]，这里描述的算法使用了这些文献中讨论的想法并将它们与DXR光线追踪和自适应的硬件曲面细分相结合。

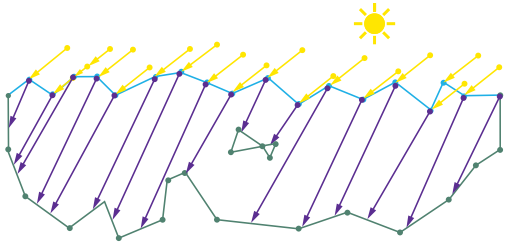
具体来说，为了渲染体积焦散，光线追踪在渲染流程中用到了两次。在初始步骤中，光线追踪被用于计算引导硬件曲面细分级别的信息，这些信息被用于自适应地切割焦散体积的三角形光束体积。用于累积向眼睛散射的体积光的渲染管线使用所有的GPU着色器阶段，例如，顶点着色器，外壳着色器，域着色器，几何着色器和像素着色器。

主要焦散图[7]包含了从光的观察点位置渲染得到的水表面的位置和法线，光线是从水面上沿着折射和反射光方向的这些位置发出的，最终在场景中相交。这些交点的​​位置存储在二级焦散图中，比如本章中将会提到的折射焦散图和反射焦散图，（主）焦散图和折射焦散图中的位置被用于定义在之后的体积切片时所使用的三角形体积光束。

**157**



**图14.1：**未受干扰的（译者注：平行）光线（译者注：穿过水体）击中水下场景。



**图14.2：**（译者注：从水面发生）折射光线（紫色）击中水下场景。

本章重点介绍折射光线的水下焦散，注意此处描述的算法也可用于渲染在水面反射并击中在水面以上的几何体的光产生的焦散。而且，也可以用任何其他透明物体的表面替代水面。

在水下游戏场景中，体积光通常是根据阴影图中编码的可见性信息生成的[4]。在这种情况下，阴影图包含了从光的位置渲染的水下的几何体。因此，它通过光栅化的方式传递了原始光线与水下场景的交点信息，见图14.1。

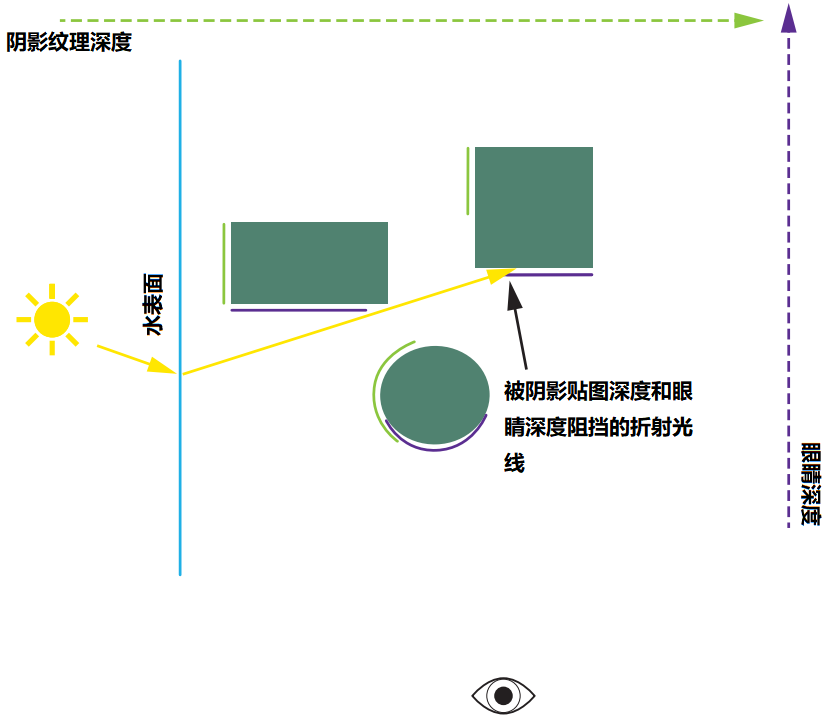
当一束光照射到水面时，如果它的能量受到水面折射改变了方向，则有必要找到折射光线与场景的交点，见图14.2。

图14.1和14.2的对比显示了所得到的交点可以是非常不同的，如果光线以较小的角度照射水面，这种差异会更加明显。折射导致光线在水下的几何体表面产生典型的焦散图案，以类似的方式，体积光也受到折射光的影响。一些发表的文章[5,6,8,9,10]描述了在焦散渲染的上​​下文中如何突破只能使用阴影贴图的限制（如图14.1所示）。

通常，使用以下两类算法中的一种：

1. 二维图像空间射线行进（大名鼎鼎的Ray march）：
   * 1. 在像素着色器中对主深度缓冲区或阴影图深度缓冲区（按一定的步长）行进（march译者注：读者需要先了解Ray march的基本原理）来找到交点，这种方法的问题在于折射光线似乎在主视图和光的视图中都被遮挡，如图14.3所示。
     2. 使用以下方法对一组图像进行渲染和行进（march）：
2. 多深度层的主深度缓冲区和阴影深度图。

**158**



**图14.3：**折射光线的交点在对光和眼睛的深度图行进（march）的过程中似乎都被阻挡了。

1. 主深度缓冲区和阴影图的多个视点。
2. 距离替身[8]。

但请注意，这些方法会增加运行时成本和内存开销，实现的复杂性会比后面描述的基于DXR的方法明显高很多。

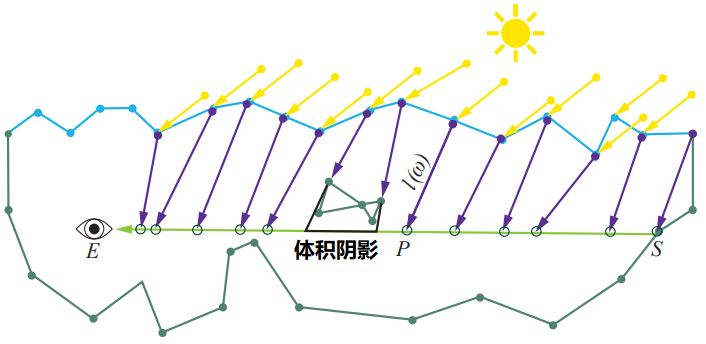
1. 三维体素网格行进（voxel grid marching）：这类算法将水下场景体素化并对所产生的网格进行行进（march），依靠网格的（高）分辨率，这些方法可以产生令人印象深刻的效果。体素化不是一个廉价的操作，可以理解为和光栅化侧是等价的，要始终保持体积边界层次结构是最新的。如果需要高网格分辨率，对内存的要求很快会变得过高。对一个充满细节的3D网格做射线行进（march）并不快，并且可能变得非常慢。总的来说，体素化方法的实现复杂度高于基于DXR的方法。

本章介绍的技术不使用任何仅是描述如何计算折射光线交点的近似方法，而是使用DXR精确地计算出折射光线击中动态的水下场景。

**14.2 体积光和折射光**

有关中间介质中的体积光计算的大致介绍，请参阅Hoobler[4]的工作。在这里，我们简单地呈现一个用双重积分描述的从水下场景点S向眼睛E

**159**



**图14.4：**图中位于左边E处的眼睛从水中看向右边的S位置。从上方（水面）照下来的光线到达这条射（视）线的时候是处在不同的位置，这取决于水表面（水面的波动状态），并将（这些照下来的）光散射到眼睛E处。

散射的辐照度L到底有多少的公式：

. (14.1)

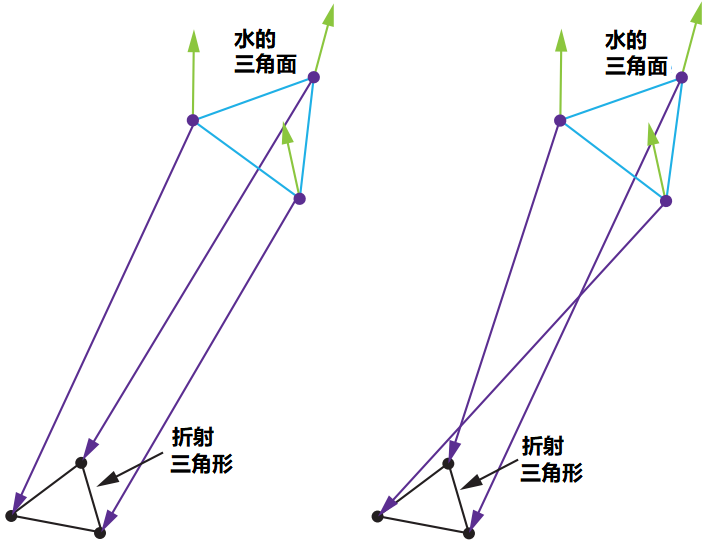
见图14.4。对于所有场景中从E开始的射线的中点P，和入射的折射光线Ω的所有方向，以下是对公式中用到的术语的说明：

1. 光线在到达P点之前沿着水下传播的消光（译者注：消光，是指光线通过溶液或物质后的透射光强度与该光线通过溶液或某一物质前的入射光强度相比，发生了衰减，即被溶液给吸收了，又称吸光）函数l加上从P点到眼睛位置E点的路径的长度（｜P－E｜），是水体积的消光系数——假定在本章剩下的部分中为常数。
2. 点P处的散射系数。
3. 相位函数决定了有多少从折射光方向照射到P点的光向眼睛散射。
4. 沿折射光方向照射到P点处的辐照度。
5. 沿折射光方向的可见度，例如折射光线是否到达了P点？

有两种可能的近似方案来对所有的散射事件进行积分计算：

1. 使用3D网格在每个网格单元的中心对散射事件进行离散的累积。  
   需要使用具有足够高分辨率的网格来防止体积光穿透薄场景（出现漏光现象）。
2. 追踪足够多的从水面上的原点发出并与水下场景有交点的折射光线。
3. 对于每一个有折射光线进入的网格单元，计算射线上最接近网格单元中心的P点。

**160**

****

**图14.5：**左：折射三角形与水面三角形成凸包。右：形成的体积发生扭曲，不是凸包。

1. 计算从P点到达眼睛的相位函数和透射辐照度。
2. 对网格单元中透射的辐照度进行累积（译者注：离散积分）。
3. 对于屏幕上的每个像素，对从像素到眼睛的光线进行追踪。沿着这条射线穿过网格并对到达眼睛的光线进行累积（译者注：离散积分）。
4. 创建一组足够密集的三角形光束体积[2]来近似使用图形管线和加算混合的散射积分。如图14.5所示，折射光方向可能会产生出非凸包的三角形光束。在第14.3节中提出了这个算法，通过尝试在区域中使用高曲面细分级别来防止这种因折射光线的方向快速变化而产生非凸包的情况。

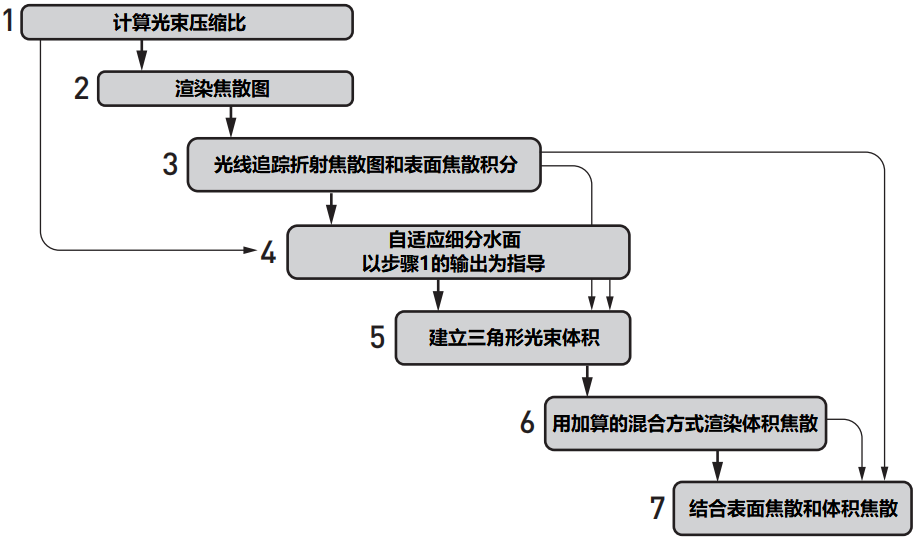
对于每个三角形光束，使用图形管线渲染八个三角形以形成光束的精确凸包围体，这些三角形是按照表面法线始终指向体积外部的方式生成的。

沿着来自眼睛的射线，折射光的方向从光线击中的点的背面三角形变为它击中的体积的正面三角形。这样的结果就是，不可能使用加算的混合方式了，在背面三角形处有一个正的散射项，在正面三角形处则有一个负的散射项，就像Golias和Jensen[3]所提出的那样。

有个可行的方法是，使用足够小的体积，通过仅在每个体积的正面三角形处累积散射项来近似散射积分。

本章附带的演示使用了加算混合，曲面细分和一个几何着色器，用于实现受到第二种方法启发的体积切片方法。这反映在以下的算法概述中。

**161**



**图14.6：**算法概述。

**14.3 算法**

演示中使用以下七个步骤来渲染体积水焦散。图14.6显示了这些步骤的概述。

请注意，如果不是沿着折射光的方向追踪光线，也可以沿着水表面的反射光的方向追踪光线，从而呈现反射体积和表面焦散。本章附带的演示也实现了除了折射的体积和表面焦散之外的反射表面焦散。

**14.3.1 计算光束压缩比**

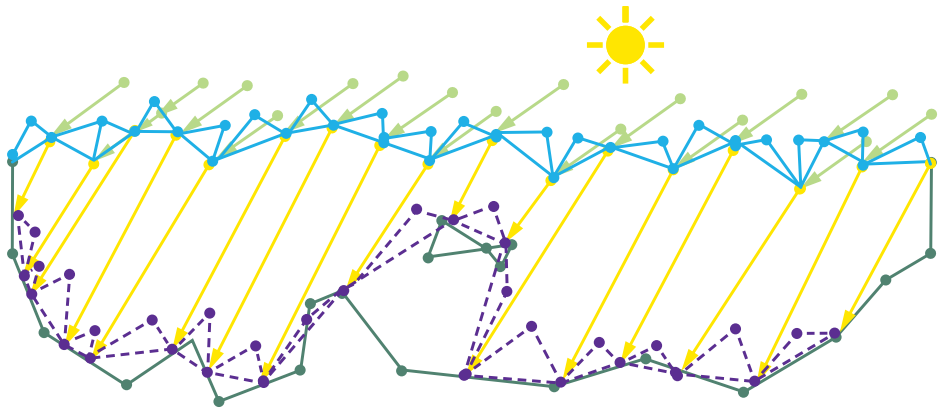
对于表示模拟水面几何的网格的每个顶点，构造一条折射光线R。这条折射光线R从当前水面的顶点的位置开始，沿着入射光的折射方向射出。

模拟折射用的水网格具有和实际水面网格完全相同数量的顶点和三角形，其顶点的位置通过每条射线R和水下的几何体的交点来计算，图14.7描述了这个过程，每一个蓝色的水表面三角形都会产生一个紫色虚线的折射水网格三角形。

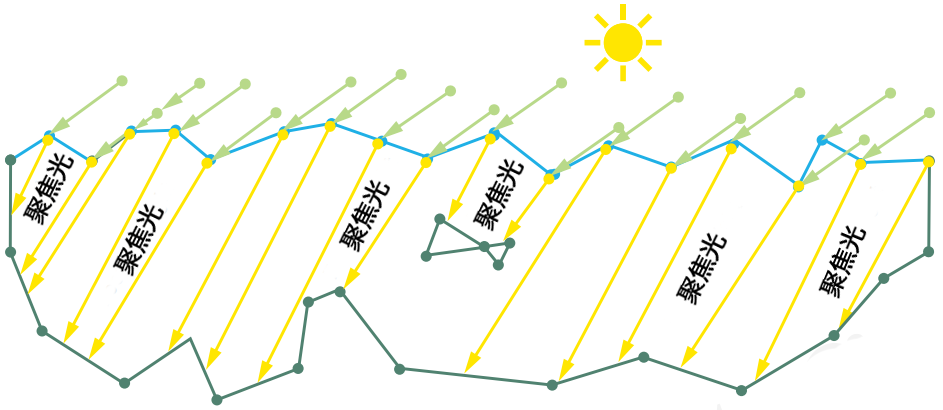
请注意，折射水网格不需要精细到能够匹配水下几何体的每一个细节，它只需要足以促进计算足够高质量的压缩比就可以了，就像下面所描述的。当水面精度不够时，该步骤会引入错误，因此，如果出现错误，则需要改善水表面（的精度）。

如图14.8所示，光线的折射可以将光

**162**



**图14.7：**计算折射水网格。



**图14.8：**光线如何聚焦在折射水网格中。

聚焦在三角形光束内，也可以相反。因此，折射水网格的三角形可以具有比它们各自对应的水面三角形更大或更小的面积。

对于每个三角形，计算光束压缩比并将其存储在一个缓存当中：

, (14.2)

其中a()计算三角形的面积，是水表面三角形，是折射三角形。

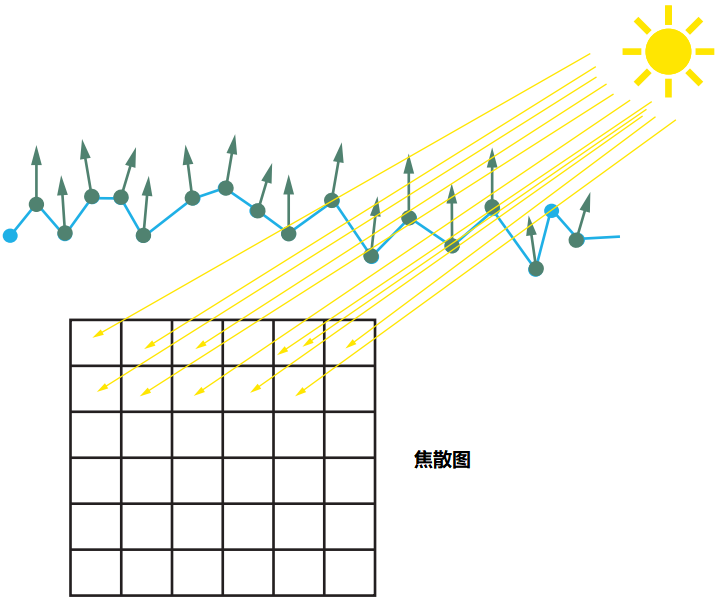
原始水三角形和折射水三角形形成粗的三角形光束，如图14.5所示，也可以考虑把压缩比作为描述三角形光束形成非凸体积的可能性。因此，压缩比可用于驱动曲面细分用于将每个粗三角形光束细分为更高密度的小光束。这个使用公式14.2的压缩比的想法并不是新的，早在之前（的资料）[3]中就已经进行了描述。

**14.3.2 渲染焦散图**

在这个步骤当中，最初将两个渲染对象清除用于表示无效的表面位置和表面法线。

接下来，所有的水面三角形使用像素着色器将以下内容写入到两个渲染对象当中：

**163**



**图14.9：**水网格被渲染成从光的角度看的焦散图——在所得表面的像素中存放了水表面的位置和法线信息。

1. 水表面的3D位置。
2. 水表面上这一点的表面法线。

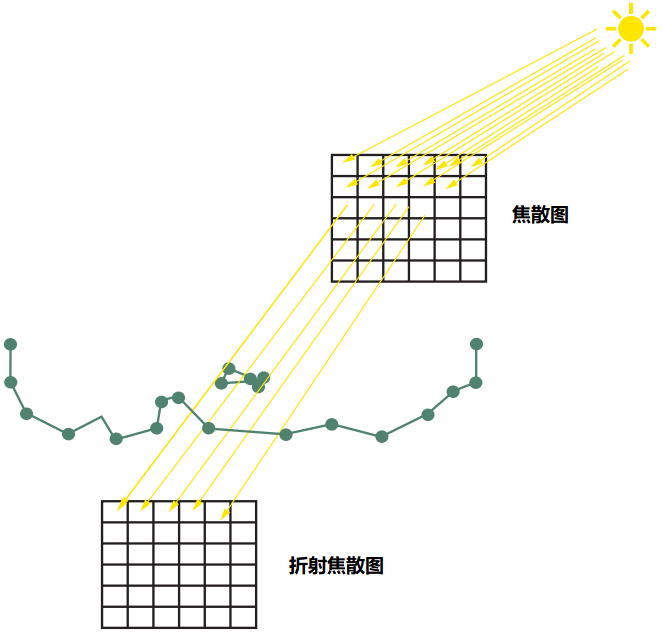
如图14.9所示。

**14.3.3 光线追踪折射焦散图和累积表面焦散**

这个步骤使用DXR为步骤2中渲染的焦散图中的有效像素进行光线追踪，与场景的交点存储在折射焦散图中。此外，交点位置被转换到屏幕空间并用于对散射表面的焦散进行累积（离散积分）：

1. 对焦散图中表示水表面上的有效点的每个像素（x，y）进行一条光线的追踪。
2. 计算光线与水下场景几何体的交点，有些情况下可以剔除掉这条光线，比如阴影图测试显示该光线被水线（水表面）以上的几何体遮住的时候。
3. 将交点的位置写入折射焦散图的像素（x，y），见图14.10。
4. 有一个可选步骤是，沿着折射焦散光线的反射方向（沿着场景法线）对二次光线进行追踪，并将得到的交点写入一次反弹焦散图的像素（x，y），见图14.11。
5. 在离屏缓冲区中对表面焦散进行积累（离散积分）。
6. 把交点（包括步骤4所说的可选点）投射到屏幕空间——如果位置在屏幕上，使用InterlockedAdd（）在缓冲区中的该屏幕位置对辐照度进行累积。为了找出交点是否对应于屏幕上最前面的像素，最简单的解决方案是进行具有一定的容错性的深度测试。

**164**



**图14.10：**对折射焦散图进行光线追踪：从存储在焦散图中的水表面位置，沿着折射光方向发射光线，并将生成的光线/场景交点存储在折射焦散图中。

另外也可以考虑用G－Buffer中存放的屏幕像素的法线和/或用深度差异的函数来缩放亮度值，也可以将三角形的唯一ID渲染到G－Buffer中并将此ID与原始模型和在DXR命中着色器中提供的实例ID进行比较。

1. 累积的辐照度值可以通过几个因子来缩放，包括来自步骤2的压缩比和/或光线穿过水[1]时按距离被吸收掉的光量。

**14.3.4 自适应地对水表面的三角形进行曲面细分**

有关三角形光束体积的自适应曲面细分的描述，请参见图14.12。

光束压缩比（见公式14.2）用于计算位于三角形光束顶部的水三角形的曲面细分因子，这个曲面细分因子缩放为：

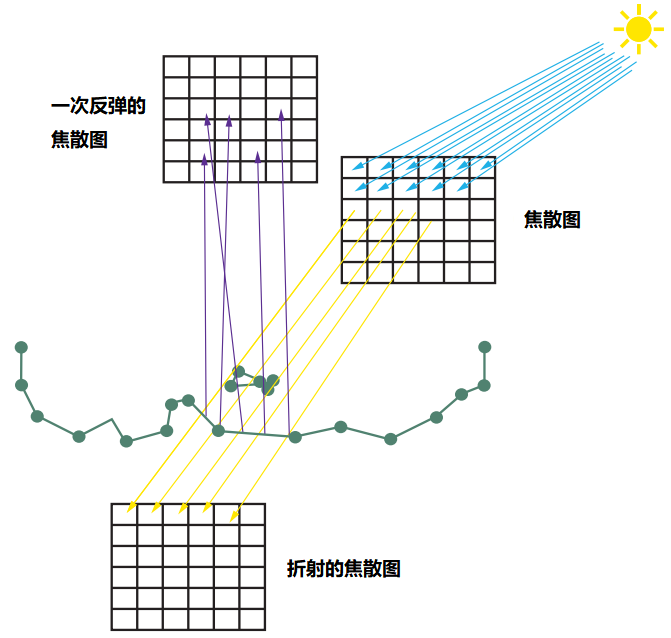
1. 为尽量接近散射积分的理想结果提供足够多的切片。
2. 防止三角形光束变成非凸包，参见图14.5。
3. 确保没有体积光穿透场景中的小部件发生漏光。

**14.3.5 构建三角形光束体积**

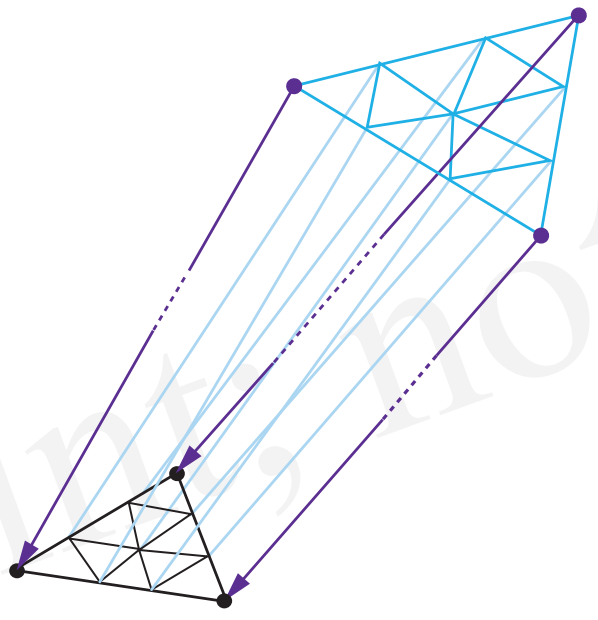
运行几何着色器来拾取经过了曲面细分的水三角形并构建出三角形光束相对应的外壳。

1. 将输入三角形的3D顶点投影到（折射）焦散图空间。

**165**



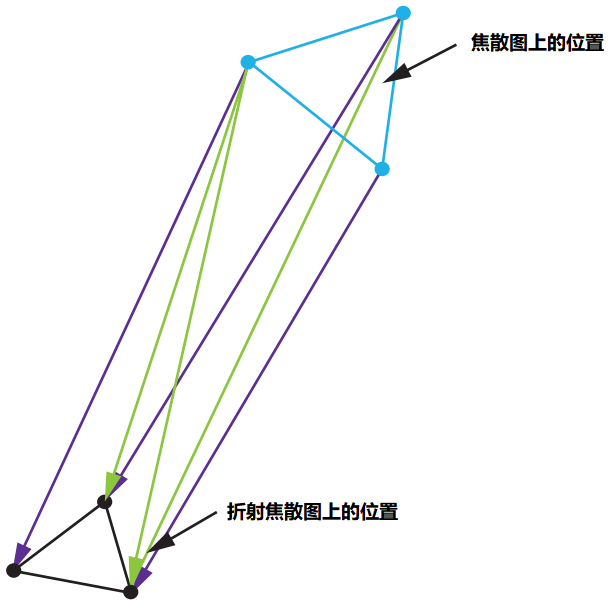
**图14.11：**为了光的另一次反弹，沿着焦散射线的反射方向进行光线追踪，创建一个一次反弹焦散图。



**图14.12：**对水三角形做自适应的曲面细分产生了细分的三角形光束——见步骤5。

1. 从焦散图读取形成体积顶盖的三角形的3D位置。
2. 从折射焦散图读取形成体积底盖的三角形的3D位置。
3. 构建形成包围体的八个三角形，见图14.13。（可选）对由折射焦散图和一次反弹焦散图创建的体积执行相同的操作。
4. 计算每个输出顶点处的三角形光束的估计厚度——这样，厚度就会通过插值传递给顶点着色器。
5. 在每个输出顶点处计算光线方向——这样，光方向就会通过插值传递给像素着色器。

**166**



**图14.13：**形成三角形光束的三角形。

**14.3.6 使用加算混合方式渲染体积焦散**

将散射光在像素着色器中以加算的方式混合到每个体积前侧的像素上并渲染到渲染对象上。

1. 在给定的经过插值的光线方向的当前3D位置处计算相位函数。
2. 将得到的散射项乘以插值的厚度。
3. 输出结果。

**14.3.7 结合表面焦散和体积焦散**

此步骤把由表面焦散照亮的场景图像和使用加算混合方式渲染的体积焦散的模糊版本进行了结合。

1. 对步骤3中的表面焦散进行模糊/降噪。
2. 使用降噪的表面焦散缓冲（图）对场景进行照亮，例如，通过乘以G－buffer像素中存储的反照率（albedo）纹理，然后加上未进行光照的结果以产生经过了光照的G－buffer。
3. 对步骤6的结果稍微进行模糊并将其添加到经过了光照的G－buffer中。

**14.4 实现细节**

如第14.1节所述，DirectX 12的DXR API用于实现所有光线追踪工作。对于步骤1，调用DispatchRays（）以便每个线程对一条折射光线在场景中准确地进行追踪，将产生的折射水网格写入一个缓冲区，该缓冲区由后面的步骤读取并使用和原来的水网格相同的索引缓冲区。

步骤2是一个普通的光栅化渲染批次的实现，对于第3步，调用DispatchRays（）从步骤2生成的焦散图的每个有效像素投射光线，可选地，着色器沿着反射方向投射额外的光线，用于由水表面反射产生的光线或一次反弹焦散图的表面焦散。

**167**

在半分辨率缓冲区中进行折射/反射光的累积，以便于快速降噪。

如果选择了额外的焦散反弹，则在步骤2中投射另一条光线模拟场景中焦散光线的反射。由此产生这些反射光线的交点被用于模拟表面焦散的间接光照并被写入另一个焦散图，即反射的焦散图——缓冲区的尺寸为了便于给这种额外的反弹绘制体积光束而指定。

在步骤6中，在半分辨率缓冲区中累积体积焦散以加速三角形光束的绘制。步骤5中的几何着色器为主要的折射焦散以及可选的记录在一次反弹焦散图中的额外反弹创建三角形光束。

步骤7中的表面焦散缓冲器的降噪通过一组交叉双边模糊步骤的迭代来完成，解释了视空间深度、法线和位​​置的差异。最后，表面焦散和体积焦散是双边升采样（降采样的反义词 upsample）的并与渲染场景相结合。

**14.5 结果**

表14.1显示了四个不同摄像机位置和光设置在一个场景中取得的焦散工作负载时间，在 NVIDIA RTX 2080 Ti的板子上以1920×1080的分辨率运行的工作负载，使用官方的DirectX 12中的DXR API。

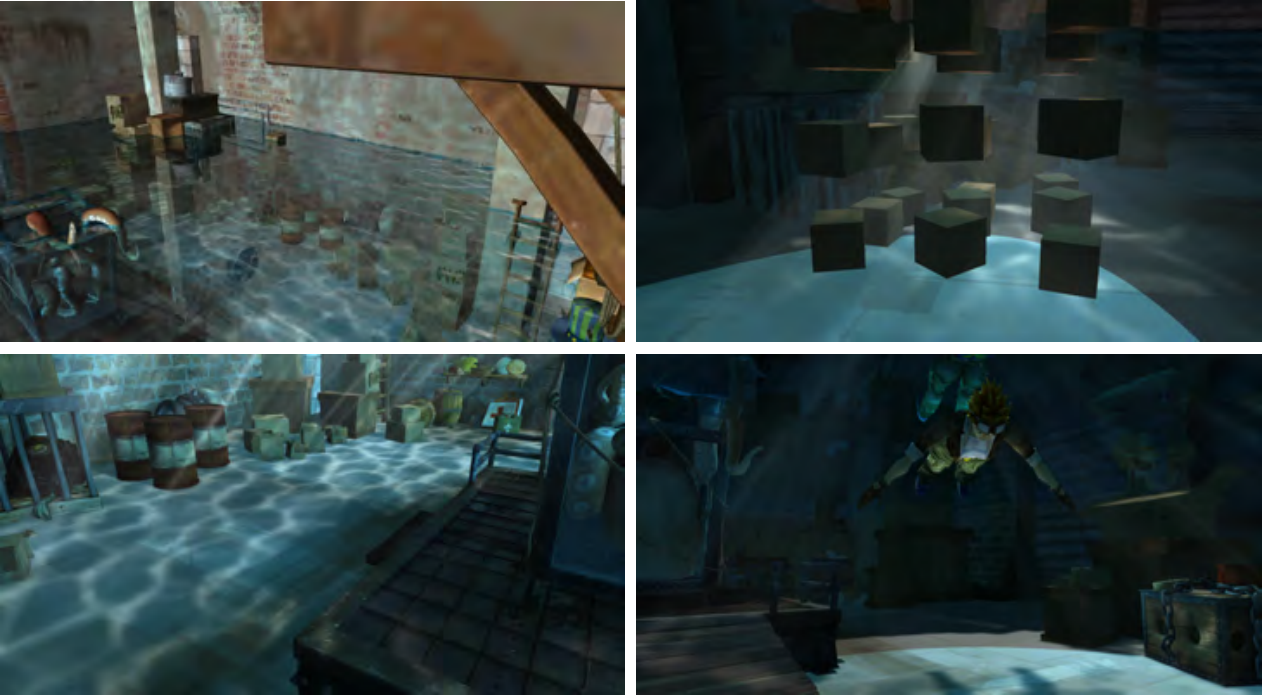
这四个场景的截图如图14.14所示，所有的场景都以超过60 FPS的交互帧速率运行，从一张2048×2048的焦散图的像素投射光线。表14.1中的时间表明，体积焦散操作在大多数情况下，能以融入现代电脑游戏可接受的时间范围内运行。相比之下，Liktor和Dachsbacher [6]的工作未能达到整合到游戏中可行的性能水平。

* 图14.14中的左上方屏幕截图显示了水面上方的视图，在这个截图中，折射体积水下焦散和在水线（面）以上可见的反射焦散是由本章中描述的算法产生的。此图像的焦散工作负载的总时间为2.9毫秒。
* 图14.14中的右上方屏幕截图显示了水面下方的视图。对于这个场景，渲染了折射体积水下焦散和二次体积反弹光。对于这种情况，体积反弹和高精度的最大曲面细分因子预设导致焦散渲染的体积部分的时间爬升到了4.6毫秒。这些设置目前在游戏中使用还是太费（昂贵）了。
* 图14.14中的左下方屏幕截图再次显示了水面下方的视图。对于这个场景，渲染了再次折射体积水下焦散和二次体积反弹光。对于这种情况，第二次体积反弹以及适当高精度的最大曲面细分因子预设导致焦散渲染的体积部分的时间最终爬到了更折中的2.1毫秒。

**168**

这些设置在一个专注于高品质体积焦散的游戏中可能是可以接受的。

* 图14.14右下方的屏幕截图显示了水面下面的另一个视图。对于这种情况，第二个体积反弹与适当高精度的最大曲面细分因子预设使得焦散渲染的体积部分的时间仅需1.4毫秒。请注意第二次反弹的光线使如何投射到角色朝下的部分的。



**图14.14：**截图。

**169**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Timings |  | |
| 截图 | 负载 | | | 时间（毫秒） |
| 左上 | 折射+反射焦散 | | |  |
|  | DispatchRays() | | | 0.9 |
|  | 表面焦散去噪 | | | 0.8 |
|  | 体积切片和升级 | | | 1.2 |
| 右上 | 折射+反射焦散+一次反弹 | | |  |
|  | DispatchRays() | | | 3.0 |
|  | 表面焦散去噪 | | | 0.8 |
|  | 体积切片和升级 | | | 4.6 |
| 左下 | 折射+反射焦散+一次反弹 | | |  |
|  | DispatchRays() | | | 0.9 |
|  | 表面焦散去噪 | | | 0.8 |
|  | 体积切片和升级 | | | 2.1 |
| 右下 | 折射+反射焦散+一次反弹 | | |  |
|  | DispatchRays() | | | 2.1 |
|  | 表面焦散去噪 | | | 0.8 |
|  | 体积切片和升级 | | | 1.4 |

**表格14.1：**计时。所有DispatchRays()都包含散射累积。

**14.6 将来的工作**

在当前的演示（demo）实现中，焦散图和折射焦散图需要有足够高的分辨率，以捕捉到水下几何体足够多的细节。调查Wyman和Nichols [10]或Liktor和Dachsbacher [6] 的想法会很有趣，可用于自适应地发射光线。

接下来，相比使用光栅化管线对集中了光的水体进行部分切片，可以更快地在体积贴图中对散射光进行累积。对于光线到眼睛的位置，信息存储在焦散图中，折射焦散图可用于预防体积光通过场景中薄的地方时发生光泄漏。

**14.7 演示**

可以在NVIDIA GPU上运行的展示了所提出的技术的演示在代码库中有提供。

**参考**

1. Baboud, L., and Decoret, X. ´ Realistic Water Volumes in Real-Time. In *Eurographics Conference on Natural Phenomena* (2006), pp. 25–32.

**170**

1. Ernst, M., Akenine-Moller, T., and Jensen, H. W. ¨ Interactive Ren-dering of Caustics Using Interpolated Warped Volumes. In *Graphics Interface* (2005), pp. 87–96.
2. Golias, R., and Jensen, L. S. Deep Water Animation and Rendering. https://www.gamasutra.com/view/feature/131445/deep\_water\_ animation\_and\_rendering.php, 2001.
3. Hoobler, N. Fast, Flexible, Physically-Based Volumetric Light Scattering. https://developer.nvidia.com/sites/default/files/akamai/gameworks/downloads/papers/NVVL/Fast\_Flexible\_Physically-Based\_Volumetric\_Light\_Scattering.pdf, 2016.
4. Hu, W., Dong, Z., Ihrke, I., Grosch, T., Yuan, G., and Seidel, H.P. Interactive Volume Caustics in Single-Scattering Media. In *Symposium on Interactive 3D Graphics and Games* (2010), pp. 109–117.
5. Liktor, G., and Dachsbacher, C. Real-Time Volume Caustics with Adaptive Beam Tracing. In *Symposium on Interactive 3D Graphics and Games* (2011), pp. 47–54.
6. Shah, M. A., Konttinen, J., and Pattanaik, S. Caustics Mapping: An Image-Space Technique for Real-Time Caustics. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (March 2007), 272–280.
7. Szirmay-Kalos, L., Aszodi, B., Laz ´ anyi, I., and Premecz, M. ´ Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum 24*, 3 (2005), 695–704.
8. Wang, R., Wang, R., Zhou, K., Pan, M., and Bao, H. An Effiffifficient GPU-based Approach for Interactive Global Illumination. *ACM Transactions on Graphics 28*, 3 (July 2009), 91:1–91:8.
9. Wyman, C., and Nichols, G. Adaptive Caustic Maps Using Deferred Shading. *Computer Graphics Forum 28*, 2 (Apr. 2009), 309–318.

**171**