用于Cel边缘渲染的算法反映了NPR的一些主要主题和技术.我们的目标是提出一种能体现域特征的算法.可以将使用的方法大致分类为基于表面着色,生成几何,图像处理,几何边缘检测或它们的混合.

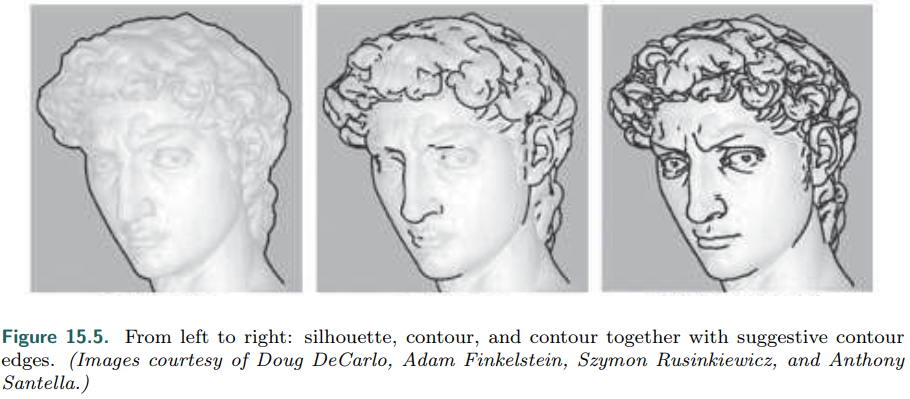
在卡通渲染中可以使用几种不同类型的边缘:

1. 边界或边缘是两个三角形不共享的边缘,例如,一张纸的边缘.固体通常没有边界边缘.
2. 折痕,硬边缘或特征边缘是两个三角形共享的边缘,并且两个三角形之间的角度(称为二面角)大于某个预定值.好的默认折痕角为60度[972].例如,一个立方体具有折边.折痕边缘可以进一步细分为山脊和山谷边缘.
3. 当两个三角形的共同边材质不同或以其它方式引起着色更改时,将出现**材质边缘**.它也可以是艺术家希望显示的边缘,例如前额线或分隔相同颜色的裤子和衬衫的线.
4. 轮廓边缘是这样的:与某个方向矢量(通常是眼睛的一个方向矢量)相比,两个相邻的三角形面向不同的方向.
5. 轮廓边缘是沿着物体轮廓的边缘,即,它在图像平面中将对象与背景分离.

参见图15.4.这种分类是基于文献中的常用用法,但是存在一些变化,例如我们所说的折痕和材料边缘有时在其他地方称为边界边缘.

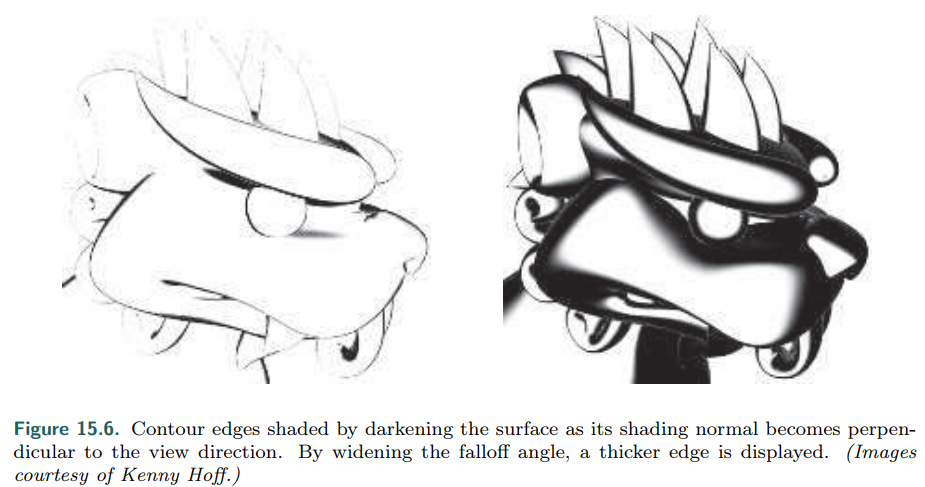
我们在这里区分轮廓和轮廓边缘.两者都是边缘,沿着该边缘,表面的一部分面向观察者,另一部分则面向外.轮廓边缘是轮廓边缘的子集，轮廓边缘将对象与另一个对象或背景分开.例如,在头部的侧视图中,即使耳朵出现在头部的轮廓轮廓内,耳朵也会形成轮廓边缘.图15.3中的其他示例包括鼻子,两个弯曲的手指以及头发所在的位置.在一些早期文献中,轮廓边缘被称为轮廓,但是通常是指轮廓边缘的全部类别.此外,轮廓线边缘不应与地形图上使用的轮廓线混淆.

请注意,边界边缘与轮廓或轮廓边缘不同.轮廓和轮廓边缘由视图方向定义,而边界边缘与视角无关.提示轮廓[335]由从原始角度看几乎是轮廓的位置形成.它们提供了有助于传达物体形状的附加边缘.参见图15.5.虽然我们的重点主要是检测和渲染轮廓边缘,但其它类型的笔划也做了很多工作[281,1014,1521].我们还主要关注于找到多边形模型的此类边.B´enard等人[132]讨论了寻找由细分曲面或其他高阶定义组成的模型轮廓的方法.



15.2.1 着色法线轮廓边缘

与第15.1节中的表面着色器类似,着色法线与眼睛方向之间的点积可用于给出轮廓边缘[562].如果该值接近零,则该表面几乎在眼睛的边缘,因此很可能在轮廓边缘附近.将此类区域涂成黑色,随着点积的增加,颜色逐渐变为白色.参见图15.6.在可编程着色器之前,该算法是通过使用带有黑环的球形环境贴图或将mipmap金字塔纹理的最顶层着色为黑色来实现的[448].今天,这种着色方式可以直接在像素着色器中实现,方法是在屏幕法线垂直于视图方向时变黑.



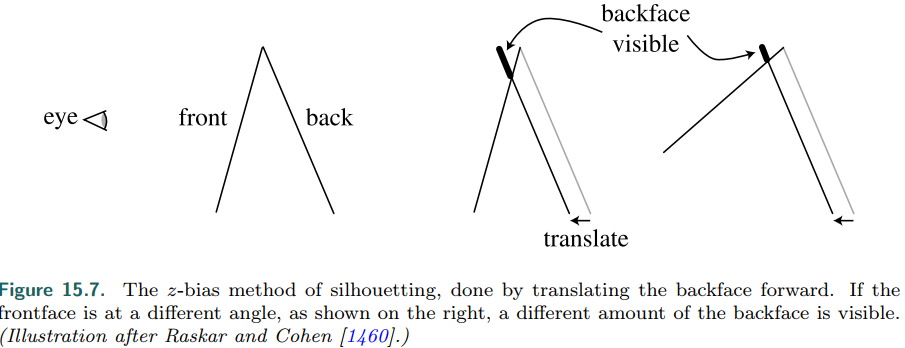
从某种意义上说,这种着色与边缘照明相反,在边缘照明中,光线照亮了对象的轮廓.在这里,场景是从眼睛所在的位置照亮的,并且衰减过大,边缘变暗.也可以将其视为图像处理中的阈值滤波器,其中,只要表面低于特定强度,图像就会转换为黑色,否则会转换为白色.

该方法的一个特点或缺点是,轮廓线的绘制宽度取决于表面的曲率.该方法适用于没有折痕边缘的曲面模型,例如轮廓上的区域通常具有法线指向几乎垂直于视图方向的像素.该算法在立方体等模型上失败,因为表面积接近 折痕边缘将不具有此属性.它也可能破裂,甚至在弯曲的表面上看起来也很糟糕,因为当物体很远并且在轮廓边缘附近采样的某些法线可能不是近乎垂直时.Goodwin等人[565]注意这个基本概念如何仍然具有作为视觉提示的有效性,并讨论了如何将照明,曲率和距离结合起来以确定笔触粗细.

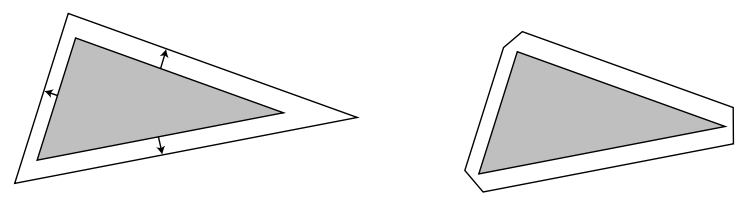
15.2.2 生成几何轮廓

Rossignac和van Emmerik [1510]提出了实时轮廓边缘渲染的最早技术之一,后来Raskar和Cohen [1460]对其进行了改进.一般的想法是正常渲染正面,然后以使轮廓轮廓可见的方式渲染背面.有多种渲染这些背面的方法,每种方法都有其优点和缺点.每种方法的第一步都是绘制正面.然后,打开正面剔除,关闭背面剔除,以便仅渲染背面.

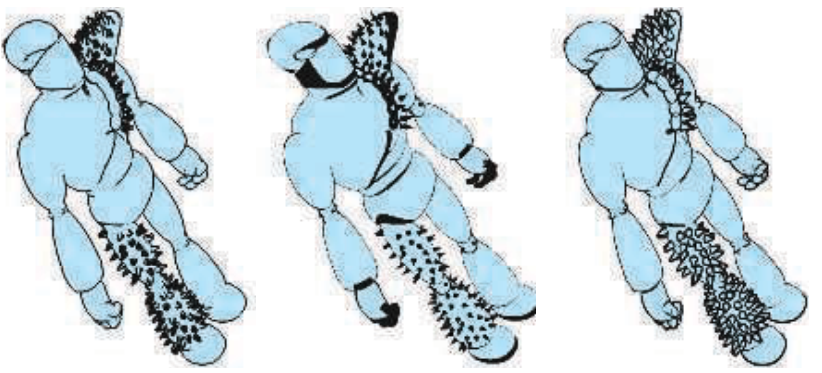
渲染轮廓的一种方法是仅绘制背面的边缘(而不是表面).使用偏斜或其他技术(第15.4节)可确保其中一些线正好在正面的前面绘制.这样,只有正面和背面相交的边缘才可见[969,1510].

 使这些线条更宽的一种方法是将背面本身变为黑色,然后再次向前偏置.Raskar和Cohen提供了几种偏置方法,例如以固定量进行平移,或者以补偿z深度的非线性性质的量进行平移,或者使用诸如OpenGL的glPolygonOffset之类的深度斜率偏置调用.Lengyel [1022]讨论了如何通过修改透视矩阵来提供更好的深度控制.所有这些方法的问题在于它们不能创建宽度一致的线.为此,向前移动的量不仅取决于背面,而且还取决于相邻的正面,参见图15.7.背面的坡度可用于使多边形向前偏置,但是线的粗细也将取决于正面的角度.

Raskar和Cohen [1460,1461]通过将每个背面三角形沿其边缘加注一定宽度以看到一致的粗线,从而解决了该邻居依赖性问题.即,三角形的斜率和与观看者的距离确定三角形扩展了多少.一种方法是将每个三角形的三个顶点沿其平面向外扩展.渲染三角形的一种更安全的方法是将三角形的每个边缘向外移动并连接边缘.这样做可以避免顶点远离原始三角形.参见图15.8.请注意,由于背面扩展到正面的边缘之外,因此此方法不需要偏置.有关这三种方法的结果,请参见图15.9.这种增肥技术更加可控且一致,并且已成功用于视频游戏中的角色概述,例如波斯王子[1138]和非洲武士[250].



**图15.8** 三角形扩增.在左侧,背面三角形沿其平面扩展.每个边缘在世界空间中移动的量不同,以使所得边缘在屏幕空间中具有相同的厚度.对于细三角形,此技术会随着一个角变长而失败.在右侧,将三角形的边缘扩展并合并以形成斜切的角,以避免出现此问题.



**图15.9** 分别使用粗线,z偏移和加宽三角形算法的背面轮廓线绘制.由于小特征的偏斜问题,背面边缘技术使线和不均匀线之间的连接不良.由于依赖于正面的角度,z-bias技术的边缘宽度不均匀.(图片由Raskar和Cohen [1460]提供.)

在刚刚给出的方法中,背面三角形沿其原始平面扩展.另一种方法是通过沿共享顶点法线移动顶点的顶点,使其背面向外移动,其角度与距眼睛的z距离成比例[671].这称为壳或光晕方法,因为偏移的背面形成围绕原始对象的壳.想象一个球体.正常渲染球体,然后以相对于球体中心5像素宽的半径扩展球体.也就是说,如果将球体的中心移动一个像素相当于在世界空间中将其移动3毫米,则将球体的半径增加15毫米.仅以黑色显示此扩展版本的背面.轮廓边缘将为5像素宽。参见图15.10。沿着顶点的法线向外移动顶点是顶点着色器的一项完美任务。这种扩展有时称为外壳映射。该方法易于实现，高效，鲁棒并且性能稳定。参见图15.11。可以通过根据背面的角度进一步扩展和着色这些底面来产生力场或光晕效果.

这种外壳技术有几个潜在的陷阱.想象一下，正面朝一个立方体看，以便只看到一张面.形成轮廓边缘的四个背面中的每个背面都将沿其对应的立方体面的方向移动,因此在拐角处留有间隙. 发生这种情况的原因是,尽管在每个角上只有一个顶点,但是每个面都有一个不同的顶点法线.问题在于,由于每个角顶点沿不同的方向扩展,因此扩展的多维数据集无法真正形成壳.一种解决方案是强制相同位置的顶点共享单个新的平均顶点法线.另一种技术是在折痕处创建退化的几何形状,然后将其扩展为具有面积的三角形.里拉等人[1053]使用附加的阈值纹理来控制每个顶点移动了多少.

壳和扩增技术浪费了一些填充物,因为所有背面都沿着管道输送.所有这些技术的其他局限性在于,几乎无法控制边缘外观,并且半透明表面很难正确渲染,具体取决于所使用的透明度算法.

整个几何技术类别的一个有价值的功能是,在渲染期间不需要连接信息或边缘列表.每个三角形均与其余三角形无关地进行处理,因此此类技术适用于GPU实现[1461].

此类算法仅渲染轮廓边缘.Raskar[1461]提供了一个聪明的解决方案,无需在变形模型上绘制脊线折痕边缘,而无需创建和访问边缘连通性数据结构.这个想法是沿着要渲染的三角形的每个边缘生成一个附加的多边形.这些边缘多边形通过用户定义的临界二面角弯曲而远离三角形的平面,该角度决定了何时应可见折痕.如果在任何给定的时刻，两个相邻的三角形都大于此折角，则边缘多边形将可见,否则它们将被三角形隐藏.参见图15.12.谷底边缘可以通过类似的技术来实现,但是需要模板缓冲区和多次通过.

15.2.3 通过图像处理进行边缘检测

上一节中的算法有时会归类为基于图像的算法,因为屏幕分辨率决定了它们的表现方式.另一类算法更直接基于图像,因为它完全对存储在图像缓冲区中的数据进行操作,并且不修改(甚至不直接知道)场景中的几何形状.

Saito和Takahashi [1528]首先介绍了这种G缓冲区概念,该概念也用于延迟着色(第20.1节).Decaudin [336]扩展了G缓冲区的使用来执行卡通渲染.基本思想很简单:可以通过对各种信息缓冲区执行图像处理算法来完成NPR.通过寻找相邻的z缓冲区值中的不连续性,可以找到许多轮廓边缘位置.相邻表面法线值的不连续性通常表示轮廓和边界边缘的位置.以环境颜色或带有对象标识值渲染场景可用于检测材质,边界和真实轮廓边缘.

检测和渲染这些边缘包括两部分.首先,渲染场景的几何图形,并使用像素着色器保存深度,法线,对象ID或其他渲染目标所需的其他数据.然后以与第12.1节中所述类似的方式执行后处理过程.后处理通道对每个像素周围的邻域进行采样,并根据这些采样输出结果.例如,假设我们对场景中的每个对象都有唯一的标识值.在每个像素处,我们可以对该ID进行采样,并将其与测试像素角落处的四个相邻像素ID值进行比较.如果任何ID与测试像素的ID不同,则输出黑色,否则输出白色.对所有八个相邻像素进行采样更加安全,但采样成本更高.这种简单的测试可用于绘制大多数对象的边界和轮廓边缘(真实轮廓).材料ID可用于查找材料边缘.

轮廓边缘可以通过在普通和深度缓冲区上使用各种过滤器来找到.例如,如果相邻像素之间的深度差高于某个阈值,则轮廓边缘可能存在,因此将像素设为黑色.与其简单地判断相邻像素是否与我们的样本相匹配,还需要其他更精细的边缘检测算子.我们不会在这里讨论各种边缘检测滤波器的优缺点,例如Roberts cross，Sobel和Scharr,因为图像处理文献广泛地介绍了这些问题[559,1729].由于此类运算符的结果不一定是布尔值,因此我们可以在某些区域中调整其阈值或在黑白之间淡入淡出.请注意,法线缓冲区还可以检测折痕边缘,因为法线之间的较大差异可以表示轮廓或折痕边缘.Thibault和Cavanaugh[1761]讨论了他们如何将这种技术与深度缓冲区一起用于《无主之地》游戏.除其他技术外,他们还修改了Sobel滤波器,以便它创建单像素宽的轮廓线和深度计算以提高精度.参见图15.13.也可以沿另一个方向移动,通过忽略相邻深度差异很大的边缘而仅在阴影周围添加轮廓[1138].

扩张算子是一种形态算子,用于加厚检测到的边缘[226,1217].生成边缘图像后,将应用单独的Pass.在每个像素处,都会检查像素值及其周围值(在某个半径范围内).找到的最暗像素值将作为输出返回.这样,一条细黑线将被搜索区域的直径加粗.可以将多次Pass应用于进一步的加粗生产线,但要权衡的是,每次Pass需要的样本数量要少得多,从而抵消了额外通过的成本.不同的结果可以具有不同的厚度,例如,可以使轮廓边缘比其他轮廓边缘更厚.相关的腐蚀算子可用于细化线或其他效果.有关某些结果,请参见图15.14.

这种算法有几个优点.与大多数其他技术不同,它可以处理所有类型的平面或曲面.Mesh不必连接,甚至不必保持一致,因为该方法基于图像.

这种技术的缺陷相对较少.对于接近边缘的表面,z深度比较过滤器可能会错误地检测到整个表面上的轮廓边缘像素.z深度比较的另一个问题是,如果差异很小,则轮廓边缘可能会丢失.例如,桌上的一张纸通常会遗漏其边缘.同样,法线贴图滤镜会丢失这张纸的边缘,因为法线是相同的.这仍然不是万无一失的.例如,一张折叠到其自身上的纸将在边缘重叠处创建无法检测的边缘[725].生成的行显示阶梯状混叠,但是第5.4.2节中描述的形态学抗锯齿技术可以很好地与这种高对比度输出以及后期处理等技术配合使用,以提高边缘质量.

检测也可能以相反的方式失败,从而在不应存在的边缘上创建边缘.确定构成边缘的方法并非万无一失.例如,想象一下玫瑰,细圆柱的茎.特写.样本像素附近的茎法线变化不大,因此没有检测到边缘.随着我们远离玫瑰,法线在像素之间的变化将更快,直到由于这些差异而在某个点附近的边缘可能发生错误的边缘检测.从深度图检测边缘也会发生类似问题,而透视图对深度的影响是需要补偿的另一个因素.Decaudin[336]提供了一种通过处理法线贴图和深度图的梯度而不仅仅是值本身来寻找变化的改进方法.确定各种像素差异如何转换为颜色变化是一个经常需要针对内容进行调整的过程[250,1761].参见图15.15.

一旦生成笔划,就可以根据需要执行进一步的图像处理.由于笔划可以在单独的缓冲区中创建,因此可以单独修改它们,然后在表面上进行合成.例如,可以使用噪波功能分别使线条和表面磨损和摆动,从而在两者之间形成小的间隙并呈现手绘效果.纸张的高度场可用于影响渲染,其固体材料(例如木炭)沉积在凹凸的顶部,或水彩颜料堆积在山谷中.有关示例,请参见图15.16.

在这里,我们专注于使用几何或其它非图形数据(例如法线,深度和ID)检测边缘.自然地,图像处理技术被开发用于图像,并且这样的边缘检测技术可以应用于颜色缓冲器.一种方法称为高斯差分（DoG）,其中使用两个不同的高斯滤波器对图像进行两次处理,然后从另一个中减去一个.已经发现这种边缘检测方法对NPR产生特别令人愉悦的结果,用于生成各种艺术风格的图像,例如铅笔阴影和粉彩[949,1896,1966].

在许多NPR技术中,图像后处理操作在模拟艺术媒体(例如水彩和丙烯酸涂料)中都具有突出的特点.在这个领域有大量的研究,对于交互式应用程序来说,许多挑战在于尝试使用最少数量的纹理样本进行最多的处理.双边,均值漂移和Kuwahara过滤器可以在GPU上使用,以保留边缘和平滑区域,使其看起来就像被涂漆一样[58,948].Kyprianidis等人[949]提供了该领域的图像处理效果的全面回顾和分类.Montesdeoca等人[1237]的工作是将许多简单的技术结合到以交互速率运行的水彩效果中的一个很好的例子.用水彩风格渲染的模型如图15.17所示.

15.2.4 几何轮廓边缘检测

迄今为止给出的方法的一个问题是,边缘的样式最多只能受到限制.我们不能轻易使线条看起来是虚线,更不能说是手绘的或像笔触一样.对于这种操作,我们需要找到轮廓边缘并将其直接渲染.具有独立的独立边缘实体可以创建其他效果,例如使轮廓突然变形而网格在震动中冻结.

轮廓边缘是这样的:两个相邻的三角形中的一个面向观察者,另一个则背对着观察者.测试过程为

其中和是两个三角形法线且是眼睛到边(即,边的任意一个端点)的视角方向.为了使该测试正常进行,必须使表面始终保持定向(第16.3节).

在模型中查找轮廓边缘的蛮力方法是遍历边缘列表并执行此测试[1130].Lander [972]指出,值得进行的优化是识别和忽略平面多边形内部的边.也就是说,给定一个相连的三角形网格,如果一条边的两个相邻三角形位于同一平面上,则该边不可能是轮廓边.在一个简单的时钟模型上执行此测试,将边缘计数从444个边缘减少到256个.此外,如果该模型定义了实体,则凹形边缘永远不能是轮廓边缘.Buchanan和Sousa [207]通过重复使用每个面的点积测试,避免了对每个边缘进行单独的点积测试的需要.

从头开始检测每个帧的轮廓边缘可能会很昂贵.如果摄像机视图和对象在帧与帧之间移动不大，则可以合理地假设先前帧的轮廓边缘可能仍然是有效的轮廓边缘。 Aila和Miettinen [13]将有效距离与每个边缘相关联。该距离是观看者可以移动多远，并且轮廓边缘仍保持其状态的距离。在任何实体模型中，每个单独的轮廓始终由一条闭合曲线组成,称为轮廓环,或更恰当的说是轮廓环.对于对象边界内的轮廓,循环的某些部分可能会被遮盖.甚至实际的轮廓也可能包含几个循环,其中循环的一部分位于轮廓内或被其他表面隐藏.因此,每个顶点必须具有偶数个轮廓边[23].参见图15.18.请注意,在跟随网格边缘时，循环通常在三个维度上参差不齐，z深度会明显变化。如果需要形成更平滑曲线的边缘（例如，通过距离更改厚度[565]），则可以执行其他处理以在三角形的法线之间进行插值，以近似三角形内部的真实轮廓边缘[725，726].

从一帧到另一帧跟踪循环位置比从头开始创建循环要快.Markosian等人[1125]从一组循环开始,并使用随机搜索算法在相机移动时更新此组.当模型改变方向时,也会创建并破坏轮廓循环.Kalnins等人[848]注意,当两个循环合并时，需要采取纠正措施，否则将看到从一帧到下一帧的明显跳跃.他们使用像素搜索和“投票”算法来尝试保持帧与帧之间的轮廓一致性.

这样的技术可以大大提高性能,但可能不准确.线性方法精确但昂贵.使用相机访问轮廓边的分层方法结合了速度和精度.对于非动画模型的正交视图，Gooch等人[562]使用高斯映射的层次结构来确定轮廓边缘.Sander等人[1539]使用普通圆锥的n元树（第19.3节）.Hertzmann和Zorin [726]使用模型的双空间表示法,使他们可以在模型的边缘上施加层次结构.

所有这些显式的边缘检测方法都占用大量CPU并具有较差的缓存一致性，因为形成轮廓的边缘分散在整个边缘列表中。为了避免这些成本，顶点着色器可用于检测和渲染轮廓边缘[226]。这个想法是将模型的每个边缘作为两个三角形形成一个退化的四边形，沿着管道向下发送，两个相邻的三角形法线附着在每个顶点上。当发现边缘是轮廓的一部分时，四边形的点将移动，以使其不再退化（即可见）。然后绘制该薄的四边形鳍。该技术基于与为阴影体积创建找到轮廓边缘所描述的思想相同的思想（第7.3节）。如果几何体着色器是管道的一部分，则不需要存储这些额外的鳍四边形，而是可以随时生成它们[282，299]。天真的实现会在鳍片之间留下裂缝和缝隙，可以通过修改鳍片的形状来纠正[723，1169，1492].

15.2.5 线消隐

找到轮廓后,将渲染线条.明确找到边缘的一个优点是您可以将其样式化为笔触,绘画笔触或所需的任何其他媒介.笔画可以是基本线条,带纹理的冒名顶替者(第13.6.4节),原始元素集或其他您想尝试的东西.

尝试使用几何边缘的另一个复杂之处在于,并非所有这些边缘实际上都是可见的.用于建立z缓冲区的渲染表面可以掩盖隐藏的几何边缘,这对于简单的样式(如虚线)可能就足够了.Cole和Finkelstein[282]通过沿线本身的脊线采样z深度,将其扩展到表示线的四边形.但是,使用这些方法时,沿线的每个点都是独立绘制的,因此事先没有明确定义的开始和结束位置.对于轮廓线或其他边缘(其中线段用于定义笔触或其他连续对象),我们需要知道每个笔划何时出现以及何时消失.确定每个线段的可见性称为隐藏线渲染,其中将处理一组线段以获取可见性,并返回较小的一组(可能已剪裁的)线段.

Northrup和Markosian [1287]通过渲染对象的所有三角形和轮廓边并为每个对象分配不同的标识号来解决此问题。 读取该ID缓冲区，并从中确定可见轮廓边缘。 然后检查这些可见段是否重叠，并将它们链接在一起以形成平滑的笔划路径。 如果屏幕上的线段很短，则此方法有效，但不包括对线段本身的剪辑。 然后沿着这些重构的路径渲染风格化的笔画。 笔划本身可以以许多不同的方式进行样式化，包括锥度，耀斑，摆动，超调和淡入的效果以及深度和距离提示。 图15.19中显示了一个示例。