曲面质感就是外观和感觉-只要想一想油画的质感.在计算机图形学中,纹理化是获取表面并使用某些图像,功能或其他数据源在每个位置修改其外观的过程.例如,与其精确地表示砖墙的几何形状,不如将砖墙的彩色图像应用于由两个三角形组成的矩形.观察矩形时,彩色图像显示在矩形所在的位置.除非观察者靠近墙壁,否则缺少几何细节将不会引起注意.

但是,某些带纹理的砖墙可能会因为缺乏几何形状而无法令人信服.例如,砂浆应该是哑光的,而砖是光滑的,那么观察者会注意到两种材料的粗糙度是相同的.为了产生更令人信服的体验,可以在表面上应用第二个图像纹理.根据表面上的位置,此纹理可以更改壁的粗糙,而不是更改表面的颜色.现在,砖和灰浆的图像纹理具有颜色,而新纹理具有粗糙度值.

观看者可能会看到，现在所有的砖都是光滑的，而砂浆不是，但是请注意，每个砖的表面看起来都非常平坦.这看起来不对,因为砖的表面通常会有些不规则.通过应用凹凸贴图,可以改变砖的着色法线,以便在渲染砖时,使它们看起来并不十分平滑.这种纹理会摆动矩形原始表面法线的方向,以用于计算光照.

从掠视角看,这种颠簸的错觉可能会失真.砖块应突出在砂浆上方,挡住沙砾.即使从垂直角度来看,砖块也应将阴影投射到砂浆上.视差贴图使用纹理在渲染时似乎使平面变形,而视差遮挡贴图将射线投射到高度场纹理上以提高真实感.位移贴图通过修改形成模型的三角形高度来真正位移曲面.图6.1显示了带有颜色纹理和凹凸贴图的示例.

这些是使用越来越复杂的算法可以通过纹理解决的问题类型的示例.在本章中,将详细介绍纹理化技术.首先,介绍了纹理化过程的一般框架.接下来,我们专注于使用图像对表面进行纹理化处理,因为这是实时工作中最流行的纹理化形式.简要讨论了程序纹理,然后说明了使纹理影响表面的一些常用方法.

**6.1 纹理Pipeline** 2020年11月26日10点19分

纹理化是一种用于有效地模拟表面材料和表面变化的技术.思考纹理的一种方法是考虑单个着色像素会发生什么.如上一章所述,通过考虑材料的颜色和灯光以及其他因素来计算着色.如果存在的话，透明度也会影响样本.通过修改着色方程式中使用的值来进行纹理化.这些值的更改方式通常基于表面上的位置.因此,对于砖墙示例,根据表面位置,用砖墙图像中的相应颜色替换表面上任何点的颜色.图像纹理中的像素通常称为纹理像素,以区别于屏幕上的像素.粗糙度纹理会修改粗糙度值,而凹凸纹理会更改着色法线的方向,因此每个更改都会改变着色方程的结果.

纹理化可以通过广义纹理管线来描述.稍后将介绍许多术语,但请上心:将详细描述每条管道.

空间中的位置是纹理化过程的起点.该位置可以位于世界空间中,但通常位于模型的参照系中,因此随着模型的移动,纹理也随之移动.使用Kershaw的术语[**884**],然后在空间上的这一点应用了投影仪功能,以获得一组称为**纹理坐标**的数字,这些数字将用于访问纹理.此过程称为**映射**,这导致短语**纹理映射**.有时纹理图像本身称为**纹理贴图**,尽管这并非严格正确.

在使用这些新值访问纹理之前,可以使用一个或多个对应函数将纹理坐标转换为纹理空间.这些纹理空间位置用于从纹理中获取值,例如,它们可以是图像纹理中的数组索引以检索像素.然后,取回的值可能会通过值变换函数再次进行变换,最后,这些新值将用于修改表面的某些属性,例如材质或着色法线.图6.2详细显示了应用单个纹理的过程.管道复杂的原因是每个步骤都为用户提供了有用的控件.应该注意的是,并非每个步骤都需要.

使用此管线，当三角形具有砖墙纹理并且在其表面上生成样本时,就会发生这种情况(见图6.3).找到物体(模型)的局部参照系中的位置;比如.**然后将投影仪函数应用于此位置**.正如世界地图是将三维对象投影到二维中一样,此处的投影仪函数通常将矢量更改为二元矢量.此示例中使用的投影仪函数等效于正交投影(第2.3.1节),其作用类似于幻灯片投影仪,将砖墙图像照到三角形的表面上.要返回墙,可以将其表面上的一个点转换为一对值,范围从0到1.假设获得的值为(0.32,0.29).这些纹理坐标将用于查找此位置的图像颜色.我们的砖纹理的分辨率为256×256,因此对应函数将乘以256,得到(81.92,74.24).丢掉小数部分,在砖墙图像中发现像素(81,74),并且颜色为(0.9,0.8,0.7).纹理颜色位于sRGB颜色空间,因此,如果要在着色方程式中使用该颜色,则将其转换为线性空间,得到(0.787,0.604,0.448)(第5.6节).

**6.1.1 射影[Projector]函数** 2020年11月26日10点25分

纹理处理的第一步是获取表面的位置并将其投影到纹理坐标空间中,通常是二维空间. 建模包通常允许美工定义每个顶点的坐标.这些可以从投影仪函数或网格展开算法初始化. 美工可以以编辑顶点位置的相同方式来编辑坐标.投影仪函数通常通过将空间中的三维点转换为纹理坐标来工作.建模程序中常用的功能包括球形,圆柱形和平面投影[**141,884,970**].

其他输入可用于投影仪函数.例如,可以使用表面法线来选择将六个平面投影方向中的哪个用于该表面.面相接处的接缝处出现纹理匹配问题;Geiss[**521,522**]讨论了一种将它们融合的技术.Tarini等人[**1740**]描述了**多立方体贴图**,其中一个模型映射到一组立方体投影,而不同体积的空间映射到不同的立方体.

其他投影仪函数根本不是投影,而是表面创建和细分的隐含部分.例如,参数化曲面具有自然的值集作为其定义的一部分.参见图6.4.纹理坐标也可以根据各种不同的参数生成,例如视图方向,表面温度或其他任何可想象的参数.投影仪函数的目标是生成纹理坐标.根据位置推论得出这些只是实现这一目标的一种方法.

非交互式渲染器通常将这些投影仪函数称为渲染过程本身的一部分.单个投影仪函数可能足以满足整个模型的需要,但美术师通常必须使用工具来细分模型并分别应用各种投影仪函数[**1345**].见图6.5.

在实时工作中,通常在建模阶段应用投影函数,并将投影结果存储在顶点.这并非总是如此;有时在顶点或像素着色器中应用投影函数是有利的.这样做可以提高精度,并有助于启用各种效果,包括动画(第6.4节).某些渲染方法,例如环境映射(第10.4节),具有自己的专用投影仪函数,这些函数按像素进行评估.

球形投影(图6.4的左侧)将点投射到以某个点为中心的假想球体上.此投影与Blinn和Newell的环境映射方案(第10.4.1节)中使用的投影相同,因此第407页的公式10.30描述了此函数.这种投影方法也遇到了该部分介绍的顶点插值相同的问题.

圆柱投影计算的u纹理坐标与球形投影相同,而纹理坐标计算为沿圆柱轴的距离.此投影对于具有自然轴的对象(例如旋转表面)很有用.当表面垂直于圆柱轴垂直时会发生变形.

平面投影就像X射线束一样,沿着一个方向平行投影,并将纹理应用于所有表面.它使用正投影法(第4.7.1节).例如,这种类型的投影可用于贴花(第20.2节).

由于投影方向边缘的曲面存在严重的变形,因此美术师通常必须手动将模型分解为接近平面的部分.也有一些工具可以通过解开网格或创建一组接近最佳的平面投影来帮助最大程度地减少变形,或者可以辅助此过程.我们的目标是使每个多边形在纹理区域中享有更公平的份额,同时还要保持尽可能多的网格连接.连接性很重要,因为采样伪像可以沿着纹理的各个单独部分相遇的边缘出现.具有良好展开效果的网格也可以简化艺术家的工作[**970,1345**].**16.2.1节讨论了纹理变形如何对渲染产生不利影响.图6.6显示了用于创建图6.5中的雕像的工作空间.这种展开过程是网格参数化这一较大研究领域的一个方面.有兴趣的读者可以参考Hormann等人的SIGGRAPH课程笔记**[**774**].

纹理坐标空间并不总是二维平面.有时是三维体积.在这种情况下,纹理坐标表示为三元素矢量,其中为沿投影方向的深度.其它系统最多使用四个坐标,通常指定为 [**885**];用作齐次坐标中的第四个值.它的作用类似于电影或幻灯片投影仪,投影纹理的大小随距离而增加.例如,对于将装饰性聚光图案(称为图案)投影到舞台或其他表面上很有用[**1597**].

纹理坐标空间的另一种重要类型是方向性的,其中空间中的每个点都可以通过输入方向访问.可视化这种空间的一种方法是将其作为单位球体上的点,每个点的法线表示用于访问该位置纹理的方向.使用方向参数化的最常见纹理类型是立方体贴图(第6.2.4节).

还值得注意的是,一维纹理图像和功能有其用途.例如,在地形模型上,可以通过高度确定颜色,例如低地是绿色;山峰是白色的.线条也可以纹理化;这种用法的一种用法是将雨水渲染为一组带有半透明图像的长线.这种纹理对于从一个值转换为另一值,即作为查找表也是有用的.

由于可以将多个纹理应用于一个表面,因此可能需要定义多组纹理坐标.但是,如果应用坐标值,则思路是相同的:将这些纹理坐标插值到整个曲面上,并用于检索纹理值.但是,在插值之前，这些纹理坐标由对应函数进行转换.

**6.1.2 通讯[Corresponder]函数** 2020年11月26日11点10分

通讯函数将纹理坐标转换为纹理空间位置.它们提供了将纹理应用于表面的灵活性.对应程序功能的一个示例是使用API​​选择现有纹理的一部分进行显示.仅此子映像将在后续操作中使用.

通讯器的另一种类型是矩阵变换,可以应用于顶点或像素着色器.这样可以在表面上平移,旋转,缩放,剪切或投影纹理.如第4.1.5节所述,转换的顺序很重要.令人惊讶的是,纹理转换的顺序必须与预期的顺序相反.这是因为纹理变换实际上会影响确定查看图像位置的空间.图像本身不是要转换的对象;定义图片位置的空间正在更改.

另一类通讯程序功能控制图像的应用方式.我们知道图像将出现在处于[0,1]范围内的表面上.但是超出此范围会发生什么？通讯功能确定行为.在OpenGL中,这种类型的对应功能称为“包装模式”.在DirectX中,它称为“纹理寻址模式”.这种类型的常见通讯功能是:

1. **wrap**(DirectX),**repeat**(OpenGL)或tile-图像在整个表面上重复.在算法上,纹理坐标的整数部分被删除.此功能对于使材料的图像重复覆盖表面很有用,通常是默认设置.
2. **mirror**-图像在整个表面上重复,但在其他每个重复上都被镜像(翻转).例如,图像看起来通常从0到1,然后在1和2之间反转,然后在2和3之间正常,然后反转,依此类推.这提供了沿纹理边缘的连续性.
3. **clamp**(DirectX)或**clamp to edge**(OpenGL)-超出[0,1]范围的值将被钳制到该范围.这导致图像纹理的边缘重复.此功能对于避免在纹理的边缘附近发生双线性插值时避免意外从纹理的相对边缘进行采样[885]很有用.
4. **border**(DirectX)或**clamp to border**(OpenGL)-[0,1]外部的纹理坐标使用单独定义的边框颜色进行渲染.例如,由于纹理的边缘将与边框颜色平滑融合,因此此功能可以很好地将贴图渲染到单色表面上.

参见图6.7.可以为每个纹理轴不同地分配这些对应功能,例如,纹理可以沿轴重复并固定在轴上.在DirectX中,还有一个“mirror once”模式,该模式沿纹理坐标的零值镜像一次纹理,然后进行钳位,这对于对称贴花很有用.

重复平铺纹理是向场景添加更多视觉细节的廉价方法.但是,这种技术在重复大约三遍纹理后通常看起来并不令人信服,因为眼睛会看到图案.避免此类周期性问题的常见解决方案是将纹理值与另一个非平铺纹理组合.如在安德森[**40**]描述的商业地形绘制系统中所见,这种方法可以大大扩展.在该系统中,基于地形类型,高度,坡度和其他因素组合了多个纹理.纹理图像还与场景中放置几何模型(例如灌木和岩石)的位置有关.

避免周期性的另一种选择是使用着色器程序来实现专门的对应函数,该函数随机地重新组合纹理图案或图块.Wang帖块是这种方法的一个例子.Wang帖块集是一小组具有匹配边缘的正方形瓦片.在纹理化过程中[**1860**]随机选择瓦片.Lefebvre和Neyret[**1016**]使用相关的纹理读取和表格来实现相似类型的对应函数,以避免模式重复.

最后应用的对应函数是隐式的,它是从图像的大小导出的.通常在和的[0,1]范围内应用纹理.如砖墙示例所示,通过在该范围内将纹理坐标乘以图像的分辨率,可以获得像素位置.可以在[0,1]范围内指定值的优点**是可以交换具有不同分辨率的图像纹理,而不必更改存储在模型顶点处的值**.

**6.1.3 纹理值** 2020年11月26日11点28分

在使用对应功能生成纹理空间坐标之后,使用坐标获取纹理值.对于图像纹理,这是通过访问纹理以从图像中检索纹素信息来完成的.此过程在6.2节将详细介绍.图像纹理构成了实时工作中绝大多数纹理的使用,但是也可以使用过程功能.在程序纹理化的情况下,从纹理空间位置获取纹理值的过程不涉及内存查找,而涉及函数的计算.过程纹理将在6.3节中进一步描述.

最直接的纹理值是用于替换或修改表面颜色的RGB三元组.类似地,可以返回单个灰度值.另一种要返回的数据是,如5.5节所述.值通常是颜色的不透明度,它确定颜色可能影响像素的程度.也就是说,可以存储任何其他值,例如表面粗糙度.可以在图像纹理中存储许多其他类型的数据,这将在详细讨论凹凸贴图时看到(第6.7节).从纹理返回的值可以在使用前进行转换.这些转换可以在着色器程序中执行.一个常见的示例是将数据从无符号范围(0.0到1.0)到有符号范围(-1.0到1.0)的重新映射,这用于对存储在颜色纹理中的法线进行着色.

**6.2 图像纹理** 2020年11月26日11点31分

在图像纹理化中,将二维图像有效地附着到一个或多个三角形的表面上.我们已经完成了计算纹理空间位置的过程.现在,我们将解决在给定位置下从图像纹理获取纹理值的问题和算法.在本章的其余部分中,图像纹理将简称为纹理.另外,当我们在此处引用像素单元时,是指围绕该像素的屏幕网格单元.如第5.4.1节所述,像素实际上是显示的颜色值,该颜色值可能(并且应该为了更好的质量)受到与其关联的网格单元外部的样本的影响.

在本节中,我们特别关注快速采样和过滤纹理图像的方法.第5.4.2节讨论了混叠的问题,特别是在渲染对象边缘方面.纹理也可能存在采样问题,但是它们发生在要渲染的三角形内部.

像素着色器通过将纹理坐标值传递给诸如texture2D之类的调用来访问纹理.这些值在纹理坐标中,由对应函数映射到范围.GPU负责将此值转换为texel坐标.不同API中的纹理坐标系之间有两个主要区别.在DirectX中,纹理的左上角为(0,0),右下角为(1,1).这与存储图像数据的图像类型匹配,第一行是文件中的第一行.在OpenGL中,纹理像素(0,0)位于左下角,是DirectX的y轴翻转.像素具有整数坐标,但是我们经常要访问像素之间的位置并在其中混合.这提出了像素中心的浮点坐标是什么的问题.Heckbert[**692**]讨论了如何有两种可能的系统:截断和舍入.DirectX 9在(0.0,0.0)处定义了每个中心-这使用了舍入.这个系统有些混乱,因为在DirectX的原点,左上像素的左上角的值是(-0.5,-0.5).DirectX 10向前更改为OpenGL系统,其中texel的中心具有小数值(0.5,0.5),即截断或更准确地说是落入小数的flooring. flooring是一种更自然的系统,可以很好地映射到语言,例如,像素(5,9)为坐标定义了5.0到6.0的范围,为定义了9.0到10.0的范围.

在这一点上,一个值得解释的术语是**依赖纹理读取**,它有两个定义.第一种特别适用于移动设备.当通过texture2D或类似方法访问纹理时,只要像素着色器计算纹理坐标而不是使用从顶点着色器传入的未修改纹理坐标[**66**],就会发生依赖纹理读取.请注意，这意味着对传入的纹理坐标进行任何更改,甚至包括交换和值之类的简单操作.当着色器没有从属纹理读取时,较早的不支持OpenGL ES 3.0的移动GPU可以更高效地运行,因为可以预取纹理像素数据.这个术语的另一个较旧的定义对于早期的台式机GPU特别重要.在这种情况下,当一个纹理的坐标依赖于某些先前纹理值的结果时,就会发生依赖纹理读取.例如,一种纹理可能会更改着色法线,进而改变用于访问立方体贴图的坐标.在早期的GPU上,此类功能受到限制甚至不存在.如今,**此类读取可能会影响性能,具体取决于批处理中要计算的像素数量以及其他因素.有关更多信息,请参见第23.8节**.

GPU中使用的纹理图像大小通常为像素,其中和为非负整数.这些被称为二次幂(POT)纹理.现代GPU可以处理任意大小的非2幂(NPOT)纹理,从而可以将生成的图像视为纹理.但是,某些较旧的移动GPU可能不支持NPOT纹理的mipmapping(第6.2.2节).图形加速器对纹理大小有不同的上限.例如,DirectX 12最多允许个像素.

假设我们有一个256×256像素纹理的纹理,并且我们想将其用作正方形的纹理.只要屏幕上投影的正方形与纹理的大小大致相同,正方形的纹理看起来就几乎像原始图像.但是,如果投影的正方形覆盖的像素是原始图像包含的像素的十倍(称为放大倍数),或者投影的正方形仅覆盖屏幕的一小部分(缩小),会发生什么？答案是,这取决于您决定在这两种情况下使用哪种采样和过滤方法.

本章讨论的图像采样和滤波方法适用于从每个纹理读取的值.但是,期望的结果是防止最终渲染的图像出现混叠,这在理论上需要对最终像素的颜色进行采样和过滤.这里的区别是过滤着色方程的输入或过滤其输出.只要输入和输出是线性相关的(对于诸如颜色的输入来说就是如此),那么过滤单个纹理值就等于过滤最终颜色.但是,许多存储在纹理中的着色器输入值(例如表面法线和粗糙度值)与输出具有非线性关系.标准纹理过滤方法可能不适用于这些纹理,从而导致锯齿. 在9.13节中讨论了过滤这种纹理的改进方法.

**6.2.1 Magnification** 2020年11月26日11点35分

在图6.8中,将大小为48×48像素的纹理贴图到一个正方形上,并且就纹理尺寸而言,正方形看起来非常接近,因此基础图形系统必须放大该纹理.放大倍数最常用的滤波技术是最近邻(实际的滤波称为盒滤波器,请参见第5.4.1节)和双线性插值.还有三次卷积,它使用4×4或5×5纹素阵列的加权和.这样可以实现更高的放大质量.尽管目前尚不普遍支持三次卷积(也称为双三次插值)的本机硬件,但可以在着色器程序中执行.

在图6.8的左侧,使用了最近邻居方法.这种放大技术的一个特征是各个纹理像素可能变得明显.将该现象称为**像素化**,是因为该方法在放大时采用距每个像素中心最近的texel值,从而导致块状外观.尽管这种方法的质量有时很差,但每个像素仅需要提取一个纹理像素.

在同一图的中间图像中,使用了双线性插值(有时称为线性插值).对于每个像素,这种滤波都会找到四个相邻的纹理像素,并在二维上进行线性插值,以找到像素的混合值.结果是模糊的,使用最近邻居方法产生的许多锯齿现象已消失.作为实验,请在斜视的同时尝试看左图像,因为它的效果与低通滤镜大致相同,并且可以使脸部露出更多.

返回第170页的砖纹理示例:在不删除分数的情况下,我们获得了.我们在这里使用OpenGL的左下原点纹理坐标系,因为它与标准的笛卡尔系统相匹配.我们的目标是在四个最接近的像素之间进行插值,并使用其像素中心定义一个像素大小的坐标系.见图6.9.要找到四个最近的像素,我们从样本位置减去像素中心分数(0.5,0.5),得出.除去分数,四个最接近的像素的范围从到.在我们的示例中,分数部分(0.42,0.74)是样品相对于由四个texel中心形成的坐标系的位置.我们将此位置表示为.

将纹理访问函数定义为,其中和是整数,并返回纹理像素的颜色.可以将任意位置的双线性插值颜色计算为两步过程.首先,水平(使用)方向上对底部纹理像素和进行插值,类似地,对最顶部的两个纹理像素和.对于底部的纹理像素,我们获得(图6.9中的底部绿色圆圈),对于顶部的像素,我们获得（上方的绿色圆圈）.然后使用在垂直方向上对这两个值进行插值,因此在处的双线性插值颜色为

凭直觉,接近我们样本位置的纹素将对最终值产生更大的影响.这确实是我们在等式中看到的.在处的右上纹素具有的影响.请注意对称性:右上的影响力等于由左下角和采样点形成的矩形的面积.回到我们的示例,这意味着从该纹理像素检索的值将乘以0.42×0.74，特别是0.3108.从该texel顺时针方向，其他乘数分别为0.42×0.26、0.58×0.26和0.58×0.74,所有这四个权重的总和为1.0.

解决放大倍率模糊的常见方法是使用**细节纹理**.这些纹理代表了精细的表面细节,从手机上的划痕到地形上的灌木丛.这样的细节作为单独的纹理以不同的比例覆盖在放大的纹理上.细节纹理的高频重复图案与低频放大的纹理相结合,具有类似于使用单个高分辨率纹理的视觉效果.

双线性插值在两个方向上线性插值.但是,不需要线性插值.假设纹理由棋盘图案中的黑白像素组成.使用双线性插值可在整个纹理上提供变化的灰度样本.通过重新映射,例如,所有低于0.4的灰度都是黑色,所有高于0.6的灰度都是白色,并且将两者之间的灰度拉伸以填充间隙,纹理看起来更像棋盘,同时还使纹理像素之间有些融合.参见图6.10.

使用更高分辨率的纹理将具有相似的效果.例如,假设每个方格正方形由4×4纹素而不是1×1组成.在每个方格的中心周围,插值的颜色将完全是黑色或白色.

在图6.8的右侧,已使用了双三次过滤器,剩余的块状现象已被大大消除.应当注意,双三次滤波器比双线性滤波器更昂贵.但是,许多高阶滤波器可以表示为重复线性插值[**1518**](另请参见第17.1.1节).结果,纹理单元中用于线性插值的GPU硬件可以通过几次查找来利用.

如果双三次滤波器被认为太昂贵,Qu´ılez[**1451**]提出了一种简单的技术，即使用平滑曲线在一组2×2纹素之间进行插值.我们首先描述曲线,然后描述技术.两条常用的曲线是smoothstep曲线和五次曲线[**1372**]:

这些在许多其他情况下非常有用,在这些情况下,您需要从一个值平滑地插值到另一个值. smoothstep曲线具有的特性,并且在0和1之间平滑.五次曲线具有相同的特性,但是,即二阶导数在曲线的起点和终点也为0.两条曲线如图6.11所示.

该技术首先通过将样本乘以纹理尺寸并加0.5来计算(与公式6.1和图6.9中使用的相同).保留整数部分以备后用,并将分数存储在和中,它们在[0,1]的范围内.然后将变换为，但仍在[0,1]的范围内.最后,减去0.5,再将整数部分相加;然后,将所得的u坐标除以纹理宽度,并以v进行类似.在这一点上,新的纹理坐标与GPU提供的双线性插值查找配合使用.请注意,此方法将在每个纹理像素处产生平稳状态,这意味着,例如,如果这些纹理像素位于RGB空间中的平面上,则此类型的插值将给出平滑但仍然呈阶梯状的外观,但不一定总是想要的.参见图6.12.

**6.2.2 Minification** 2020年11月26日11点46分

当纹理最小化时,几个纹理像素可能会覆盖像素的单元,如图6.13所示.为了获得每个像素的正确颜色值,您应该整合影响像素的纹理像素的效果.但是,很难精确地确定特定像素附近所有纹理像素的确切影响,并且实际上不可能实时完美地做到这一点.

由于此限制,在GPU上使用了几种不同的方法.一种方法是使用最接近的像素,它的工作原理与相应的放大滤镜完全相同,即,它选择在像素像元中心可见的纹理像素.该过滤器可能会导致严重的混叠问题.在图6.14中,最上方的邻居被使用.朝着地平线看,由于选择了影响像素的众多纹理像素中的仅一个表示表面,因此出现了伪影.当表面相对于观看者移动时,这些伪像甚至更加明显,并且是所谓的**时间混叠**的一种表现.

经常使用的另一个过滤器是双线性插值,其工作原理与放大过滤器完全相同.该过滤器仅比最邻近的方法略胜一筹.它混合了四个纹理像素,而不是仅使用一个,但是当一个像素受到四个以上纹理像素的影响时,滤镜很快就会失效并产生锯齿.

更好的解决方案是可能的.如第5.4.1节所述,可以通过采样和过滤技术解决混叠问题.纹理的信号频率取决于其纹理像素在屏幕上的间隔距离.由于奈奎斯特限制,我们需要确保纹理的信号频率不大于采样频率的一半.例如,一个图像是由交替的黑白线组成的,相隔一个纹素.波长为两个纹理像素宽(从黑线到黑线),因此频率为.要在屏幕上正确显示此纹理,频率必须至少为,即至少一个像素/每纹理.因此,对于一般的纹理,每个像素最多应有一个纹理像素以避免混叠.

为了实现此目标,必须增加像素的采样频率或降低纹理频率.上一章讨论的抗锯齿方法提供了提高像素采样率的方法.但是,这些仅会有限地增加采样频率.为了更充分地解决这个问题,已经开发了各种纹理缩小算法.

所有纹理抗锯齿算法背后的基本思想是相同的:预处理纹理并创建数据结构,这将有助于计算像素上一组纹理像素的效果的快速近似值.对于实时工作,这些算法具有使用固定数量的时间和资源来执行的特征.以此方式,每个像素获取固定数量的样本,并进行组合以计算(可能很大)数量的纹理像素的效果.

**Mipmapping**

最流行的纹理抗锯齿方法称为mipmapping[1889].现在已生产的所有图形加速器均以某种形式实现了该功能.“Mip”在细小语言中代表multum,拉丁语在“小地方有很多东西”的意思是-将原始纹理反复过滤成较小图像的过程的好名字.

使用mipmapping最小化滤镜时,在进行实际渲染之前,使用一组较小版本的纹理增强原始纹理.纹理(零级)被下采样到原始面积的四分之一,每个新的纹理像素值通常被计算为原始纹理中四个相邻纹理像素的平均值.新的一级纹理有时称为原始纹理的子纹理.递归执行缩小,直到纹理的一个或两个尺寸等于一个纹理像素.此过程如图6.15所示.整个图像集通常称为mipmap链.

形成高质量Mipmap的两个重要元素是良好的过滤和伽马校正.形成Mipmap级别的常见方法是获取每2×2组像素,并将它们平均以得到Mip像素值.这样,使用的过滤器便是盒式过滤器,它可能是最差的过滤器之一.这可能会导致质量差,因为它会不必要地模糊低频,同时保留一些会引起混叠的高频[**172**].最好使用高斯,Lanczos，Kaiser或类似的滤波器.快速,免费的源代码可用于任务[**172,1592]**,并且某些API支持对GPU本身进行更好的过滤.在纹理边缘附近,在过滤期间必须注意纹理是重复还是单个副本.

对于在非线性空间中编码的纹理(例如大多数颜色纹理),在过滤时忽略伽玛校正将修改感知到的Mipmap级别的亮度[**173,607**].当您离对象越来越远并且使用了未经校正的Mipmap时,对象的整体外观可能会更暗,并且对比度和细节也会受到影响.因此,重要的是将此类纹理从sRGB转换为线性空间(第5.6节)并执行所有mipmap过滤,然后将最终结果转换回sRGB颜色空间进行存储.大多数API支持sRGB纹理,因此将在线性空间中正确生成mipmap,并将结果存储在sRGB中.访问sRGB纹理时,首先将它们的值转换为线性空间,以便正确执行放大和缩小.

如前所述,某些纹理与最终着色颜色具有根本上的非线性关系.尽管这通常会给过滤带来问题,但由于要过滤数百或数千个像素,因此mipmap生成对此问题特别敏感.为了获得最佳结果,通常需要专门的mipmap生成方法.此类方法在第9.13节中进行了详细说明.

构造纹理时访问此结构的基本过程很简单.屏幕像素将纹理自身上的区域包围起来.当像素区域投影到纹理上时(图6.16),它包括一个或多个纹理像素.严格来说,使用像素的单元格边界不是正确的方法,但是此处使用它来简化演示.单元外部的像素可能会影响像素的颜色.请参阅第5.4.1节.目的是大致确定多少纹理会影响像素.有两种常用的量度来计算（OpenGL称为,也称为**纹理的细节级别[texture level of detail]**）.一种方法是使用由像素单元形成的四边形的较长边缘来近似像素的覆盖范围[**1889**];另一个方法是使用四个微分和的最大绝对值作为度量[**901,1411**].每个微分是纹理坐标相对于屏幕轴的变化量的量度.例如,是一个像素的纹理值沿屏幕轴的变化量.有关这些方程式的更多信息，请参见Williams的原始文章[1889]或Flavell[**473**]或Pharr [**1411**]的文章.McCormack等人[**1160**]讨论了通过最大绝对值法引入混叠现象,并提出了一个替代公式.Ewins等[**454**]分析了几种质量相当的算法的硬件成本.

这些梯度值可用于使用Shader Model 3.0或更高版本的像素着色器程序.由于它们基于相邻像素值之间的差异,因此在受动态流控制影响的像素着色器部分中无法访问它们(第3.8节).为了在这样的部分中(例如,在循环内)执行纹理读取,必须较早地计算导数.请注意,由于顶点着色器无法访问梯度信息,因此在使用顶点纹理时,需要在顶点着色器本身中计算渐变或细节级别,并将其提供给GPU.

计算坐标的目的是确定沿mipmap的金字塔轴采样的位置.见图6.15.目标是达到奈奎斯特速率的像素与纹理像素比至少为1:1.此处的重要原理是,随着像素单元包含更多的纹理像素和的增加,将访问较小,较模糊的纹理版本.三元组用于访问mipmap.值与纹理级别相似，但是d是浮点值而不是整数值.采样高于位置的纹理级别和低于位置的纹理级别.位置用于从这两个纹理级别的每一个中检索双线性插值样本.然后根据从每个纹理级别到的距离对生成的样本进行线性插值.整个过程称为三线性插值,每个像素执行一次.

d坐标上的一个用户控件是细节偏差(LOD偏差)的水平.这是与相加的值,因此会影响纹理的相对清晰度.如果我们进一步向上移动金字塔以开始(增加d),则纹理将变得模糊.对于任何给定的纹理,良好的LOD偏差将随图像类型及其使用方式而变化.例如,开始时有些模糊的图像可能会使用负偏差,而用于纹理化的过滤效果差(混叠)的合成图像可能会使用正偏差.可以为纹理整体或像素着色器中的每个像素指定偏差.为了更好地控制,用户可以提供坐标或用于计算它的导数.

mipmapping的好处在于,访问并插值了预组合的像素集,而不是试图对影响像素的所有像素进行求和.无论缩小多少,此过程都会花费固定的时间.但是,mipmapping有几个缺陷[**473**].一个主要的是模糊.想象一下,一个像素单元在u方向上覆盖大量纹理像素,而在v方向上仅覆盖少数像素.这种情况通常发生在观看者近乎边缘地沿着带纹理的表面看时.实际上,可能需要沿着纹理的一个轴进行最小化,而沿着另一个轴进行放大.访问mipmap的效果是检索了纹理上的正方形区域.无法检索矩形区域.为避免混淆,我们选择纹理上像素单元的近似覆盖率的最大度量.这导致检索到的样本通常相对模糊.在图6.14的mipmap图像中可以看到这种效果.移到右侧距离的线显示为模糊.

**Summed-Area Table**

避免过度模糊的另一种方法是求和面积表(SAT)[312].要使用此方法,首先要创建一个数组,该数组的大小与纹理大小相同,但包含更多的颜色存储精度位(例如,红色,绿色和蓝色分别为16位或更多).在此阵列中的每个位置,必须计算并存储所有相应纹理的纹理像素的总和,该矩形由该位置和纹理像素(0,0)(原点)形成.在纹理化过程中,像素单元在纹理上的投影由矩形限制.然后访问求和面积表以确定该矩形的平均颜色,该颜色作为像素的纹理颜色传回.使用图6.17中所示的矩形的纹理坐标计算平均值.这可使用公式6.3中给出的公式完成:

在此,和是矩形的纹素坐标,而是该纹素的求和面积值.该方程的工作原理是:从右上角到原点,取整个区域的总和,然后通过减去相邻角的贡献减去面积A和B.区域C已被减去两次,因此将其添加到左下角.注意是区域C的右上角,即是边界框的左下角.

使用求和面积表的结果如图6.14所示.到达地平线的线在右边缘附近更锐利,但中间的对角交叉线仍然模糊不清.问题在于,当沿着纹理的对角线观察纹理时,会生成一个大矩形,其中许多纹理像素不位于要计算的像素附近.例如,假设一个长而细的矩形代表像素单元的反投影,该像素在图6.17的整个纹理对角线处.将返回整个纹理矩形的平均值,而不只是像素单元内的平均值.

求和面积表是所谓的各向异性过滤算法的一个示例[**691**].这样的算法在非正方形区域上检索纹理像素值.但是,SAT能够在主要水平和垂直方向上最有效地做到这一点.还要注意,求和区域表对大小为16×16或更小的纹理的占用内存至少是其两倍，而较大的纹理则需要更高的精度.

求和面积表可以在现代GPU上实现,并以合理的总体内存成本提供更高的质量[**585**].改进的过滤对于高级渲染技术的质量至关重要.例如,Hensley等人[**718，719**]提供了一种有效的实现方法,并展示了总面积采样如何改善光泽反射.可以通过SAT改进使用区域采样的其他算法,例如景深[**585,719**],阴影图[**988**]和模糊反射[**718**].

**Unconstrained Anisotropic Filtering**

对于当前的图形硬件,进一步改善纹理过滤的最常用方法是重用现有的mipmap硬件.基本思想是对像素单元进行反投影,然后对纹理上的这个四边形(四边形)进行几次采样,然后合并采样.如上所述,每个mipmap样本都有一个位置和一个与其关联的正方形区域.该算法不使用单个mipmap样本来近似该四边形的覆盖范围,而是使用几个正方形覆盖四边形.可以使用四边形的较短边来确定(与mipmapping不同,后者通常使用较长边);这会使每个Mipmap样本的平均面积变小(从而减少模糊).方形的较长边用于创建一条平行于较长边并穿过方形中间的各向异性线.当各向异性量在1：1和2：1之间时,沿着这条线采集两个样本(见图6.18).在较高的各向异性比率下,沿轴将获取更多的样本.

该方案允许各向异性线在任何方向上运行,因此没有求和面积表的限制.它也不需要比mipmaps更多的纹理内存,因为它使用mipmap算法进行采样.各向异性过滤的一个例子如图6.19所示.

Schilling等人及其Texram动态存储设备[**1564**]首先引入了沿轴采样的想法.Barkans描述了该算法在Talisman系统中的用法[**103**].McCormack等人[1161]提出了一种类似的系统,称为Feline. Texram的原始配方是沿着各向异性轴(也称为探针)给予相同权重的样品. Talisman在轴的相对两端将两个探针的去找你红减半. Feline使用高斯滤波器内核对一组探针加权.这些算法采用高质量的软件采样算法,例如椭圆加权平均(EWA)滤波器,该滤波器将像素的影响区域转换为纹理上的椭圆,并通过滤波器内核对椭圆内的纹理像素进行加权[**691**]. Mavridis和Papaioannou提出了几种在GPU上使用着色器代码实现EWA过滤的方法[**1143**].

**6.2.3 体积纹理** 2020年11月26日13点07分

图像纹理的直接扩展是通过或(值)访问的三维图像数据.例如,医学成像数据可以生成为三维网格;通过在该网格中移动多边形,可以查看这些数据的二维切片.一个相关的想法是用这种形式表示体积的灯光.通过找到其在该体积内的位置值以及光的方向,可以找到表面上某个点的照明.

大多数GPU支持对体积纹理进行mipmapping.由于在体积纹理的单个mipmap级别内进行过滤涉及三线性插值,因此在mipmap级别之间进行过滤需要四线性插值.由于这涉及对16个纹理像素的结果进行平均,因此可能会导致精度问题,可以使用更高精度的体积纹理来解决. Sigg和Hadwiger[**1638**]讨论了与体积纹理有关的此问题和其他问题，并提供了执行滤波和其他操作的有效方法.

尽管体纹理具有更高的存储要求,并且过滤成本更高,但它们确实具有一些独特的优势.由于可以将三维位置直接用作纹理坐标,因此可以跳过为三维网格找到良好的二维参数化的复杂过程.这避免了二维参数化中常见的变形和接缝问题.体积纹理也可以用于表示诸如木材或大理石的材料的体积结构.具有这种纹理的模型看起来将由这种材料雕刻而成.

由于不使用绝大多数样本,因此使用体积纹理进行表面纹理化效率极低. Benson和Davis [**133**]和DeBry等[**334**]讨论了在稀疏八叉树结构中存储纹理数据.该方案非常适合交互式三维绘画系统,因为在创建时不需要为表面分配明确的纹理坐标,并且八叉树可以将纹理细节保持在所需的任何级别. Lefebvre等人[**1017**]讨论了在现代GPU上实现八叉树纹理的细节. Lefebvre和Hoppe[**1018**]讨论了将稀疏体积数据打包为明显较小的纹理的方法.

**6.2.4 Cube Maps**

另一种纹理类型是立方体纹理或立方体贴图,它具有六个正方形纹理,每个纹理都与一个立方体的一个面相关联.使用三元纹理坐标向量访问立方体贴图,该向量指定了从立方体中心向外指向的光线的方向.光线与立方体相交的点如下.具有最大幅度的纹理坐标选择相应的面(例如,矢量选择-z面).其余两个坐标除以最大幅度坐标的绝对值,即8.4.它们现在的范围是-1到1,并被简单地重新映射到[0,1]以便计算纹理坐标.例如,坐标映射为（（-3.2 / 8.4 + 1）/ 2，（5.1 / 8.4 + 1）/ 2）≈（0.31，0.80）.立方体贴图对于表示作为方向函数的值很有用;它们最常用于环境映射(第10.4.3节).

**6.2.5 纹理表达**

处理应用程序中的许多纹理时,有几种方法可以提高性能.纹理压缩在第6.2.6节中介绍,而本节的重点是纹理地图集,纹理阵列和无绑定纹理,所有这些目的都是为了避免在渲染时更改纹理的成本.在19.10.1和19.10.2节中,描述了纹理流和转码.

为了能够为GPU分配尽可能多的工作,通常最好尽可能少地更改状态(第18.4.2节).为此,可以将多个图像放入一个较大的纹理中,称为**纹理集[texture atlas]**.这在图6.20的左侧说明.注意,子纹理的形状可以是任意的,如图6.6所示. Néoll和Stri1cker[**1286**]描述了子纹理放置集的优化.由于mipmap的上层可能包含几个单独的,不相关的形状,因此也需要注意mipmap的生成和访问.Manson和Schaefer[**1119**]提出了一种通过考虑曲面的参数化来优化mipmap创建的方法,该方法可以产生明显更好的结果. Burley和Lacewell[213]提出了一个称为Ptex的系统,其中细分曲面中的每个四边形都有自己的小纹理.优点是,这避免了在网格上分配唯一的纹理坐标,并且在纹理图集的不连续部分的接缝处没有伪影.为了能够跨四边形进行过滤, Ptex使用了邻接数据结构.当最初的目标是生产渲染时, Hillesland [746]提出了打包的Ptex,它将每个面的子纹理放入纹理地图集中,并使用相邻面的填充来避免过滤时的间接性. Yuksel [1955]提出了网格颜色纹理,该纹理在Ptex上有所改进. Toth [1780]通过实现一种方法,在过滤器抽头超出的范围时将其丢弃,从而为类似Ptex的系统提供了跨面的高质量过滤.

使用图集的一个困难是环绕/重复和镜像模式,这不会适当地影响子纹理,而只会影响整个纹理.为图集生成Mipmap时,可能会发生另一个问题,其中一个子纹理会渗入另一个子纹理.但是,可以通过在将每个子纹理放入大型纹理图集之前分别为它们生成mipmap层次结构,并对子纹理使用二次幂分辨率来避免这种情况[1293].

解决这些问题的一个更简单的解决方案是使用一种称为纹理数组的API构造,该构造完全避免了mipmapping和重复模式的任何问题[**452**].参见图6.20的右侧.纹理数组中的所有子纹理都必须具有相同的尺寸,格式,mipmap层次结构和MSAA设置.就像纹理集一样,仅对纹理数组执行一次设置,然后可以使用着色器中的索引访问任何数组元素.这比绑定每个子纹理[452]快5倍.

API还支持**无绑定纹理[bindless textures][1407]**,这也可以帮助避免状态更改成本.如果没有无绑定纹理,则使用API将纹理绑定到特定的纹理单元.一个问题是纹理单元数量的上限,这会使程序员感到麻烦.驱动程序确保纹理位于GPU端.使用无绑定纹理时,纹理数量没有上限,因为每个纹理仅由一个64位指针(有时称为“句柄”)与其数据结构相关联.可以通过许多不同方式来访问这些句柄,例如,通过制服,通过变化的数据,来自其他纹理或从着色器存储缓冲区对象(SSBO).应用程序需要确保纹理驻留在GPU端.无绑定纹理避免了驱动程序中的任何类型的绑定成本,从而使渲染速度更快.

**6.2.6 纹理压缩** 2020年11月26日13点21分

直接攻击内存和带宽问题以及缓存问题的一种解决方案是固定速率纹理压缩[**127**].通过让GPU即时解码压缩的纹理,纹理可以需要更少的纹理内存,因此可以增加有效的缓存大小.至少同样重要的是,这种纹理的使用效率更高,因为它们在访问时消耗的内存带宽更少.一个相关但不同的用例是添加压缩以提供更大的纹理.例如,在分辨率下每texel使用3字节的非压缩纹理将占用768 kB.使用纹理压缩时,压缩比为6:1,纹理将仅占用512 kB.

在图像文件格式(例如JPEG和PNG)中使用了多种图像压缩方法,但是在硬件中实现这些方法的解码成本很高(尽管有关纹理代码转换的信息,请参见第19.10.1节).S3开发了一种称为S3纹理压缩（S3TC）[**1524**]的方案,该方案被选作DirectX的标准,并称为DXTC.在DirectX 10中,它称为BC(用于块压缩).此外,它是OpenGL中的事实上的标准,因为几乎所有GPU都支持它.它具有创建大小固定的压缩图像,具有独立编码的片段并且易于解码(因此快速)的优点.图像的每个压缩部分都可以独立处理.没有共享的查找表或其他依赖项,从而简化了解码.

DXTC/BC压缩方案有七个变体,它们具有一些共同的属性.编码是在4×4 texel块(也称为图块)上完成的.每个块被单独编码.编码基于插值.对于每个编码量,存储两个参考值(例如,颜色).将为该块中的16个纹理像素的每一个保存一个插值因子.它沿着两个参考值之间的直线选择一个值,例如,等于或从两种存储的颜色内插的颜色.压缩来自仅存储两种颜色以及每个像素较短的索引值.

准确的编码在这七个变体之间有所不同,表6.1对此进行了总结.请注意,“DXT”表示DirectX 9中的名称,“BC”表示DirectX 10及更高版本中的名称.从表中可以看出,BC1具有两个16位参考RGB值(5位红色,6个绿色,5个蓝色),每个纹理像素具有2位插值因子,可从参考值之一或两个中间值中进行选择值.与未压缩的24位RGB纹理相比,它表示6:1的纹理压缩率.BC2以与BC1相同的方式对颜色进行编码,但为量化(原始)alpha值每纹理像素(bpt)添加4位.对于BC3,每个块都有以与DXT1块相同的方式编码的RGB数据.此外,使用两个8位参考值和每个像素3位插值因子对alpha数据进行编码.每个纹理像素可以选择参考alpha值之一或六个中间值之一.BC4具有一个通道,在BC3中编码为alpha. BC5包含两个通道,每个通道都像BC3中那样进行编码.

BC6H适用于高动态范围(HDR)纹理,其中每个纹理像素最初每个R,G和B通道都有16位浮点值.此模式使用16个字节,导致8 bpt.它具有用于单行的一种模式(类似于上述技术),以及用于两行的另一种模式,其中每个块可以从一小组分区中进行选择.两种参考颜色也可以进行增量编码以获得更高的精度,并且根据所使用的模式,它们也可以具有不同的精度.在BC7中,每个块可以包含一到三行,并存储8 bpt.目标是8位RGB和RGBA纹理的高质量纹理压缩.它与BC6H共享许多属性,但是是LDR纹理的格式,而BC6H是HDR的格式.请注意,在OpenGL中,BC6H和BC7分别称为BPTC FLOAT和BPTC.这些压缩技术可以应用于立方体或体积纹理以及二维纹理.

这些压缩方案的主要缺点是它们是有损的.也就是说,通常无法从压缩版本中检索原始图像.对于BC1-BC5,仅使用四个或八个插值来表示16个像素.如果图块中包含更多数量的不同值,则会有一些损失.实际上,如果正确使用,这些压缩方案通常会提供可接受的图像保真度.

BC1-BC5的问题之一是,用于块的所有颜色都位于RGB空间中的直线上.例如,红色,绿色和蓝色不能在单个块中表示.BC6H和BC7支持更多的线路,因此可以提供更高的质量.

对于OpenGL ES,选择了另一种称为爱立信纹理压缩(ETC)[1714]的压缩算法以包含在API中.该方案具有与S3TC相同的功能,即快速解码,随机访问,无间接查找和固定速率.它将4×4像素的块编码为64位,即每个texel使用4位.基本思想如图6.21所示.每个2×4块（或4×2块，取决于提供最佳质量的块）存储基色.每个块还从一个小的静态查找表中选择一组四个常量,并且块中的每个texel可以选择添加此表中的值之一.这修改了每个像素的亮度。图像质量与DXTC相当.

在OpenGL ES 3.0中包含的ETC2 [1715]中,未使用的位组合用于向原始ETC算法添加更多模式。未使用的比特组合是压缩表示（例如64位），其被解压缩为与另一压缩表示相同的图像。例如，在BC1中，将两个参考颜色设置为相同是没有用的，因为这将指示恒定的色块，只要一种参考颜色包含该恒定色，则可以依次获得该恒定的色块。在ETC中，一种颜色也可以从第一种颜色加上带符号的数字进行增量编码，因此计算可能会上溢或下溢。这种情况被用来表示其他压缩模式.ETC2添加了两个新模式，每个块具有四种颜色（以不同方式派生），最终模式是RGB空间中的一个平面，用于处理平滑过渡。爱立信Alpha压缩（EAC）[1868]用一个分量（例如Alpha）压缩图像。这种压缩类似于基本的ETC压缩，但仅用于一个分量，因此，生成的图像每个texel存储4位。可以选择将其与ETC2结合使用，此外，可以使用两个EAC通道来压缩法线（有关此主题的更多信息，请参见下文）。所有ETC1，ETC2和EAC都是OpenGL 4.0核心配置文件，OpenGL ES 3.0，Vulkan和Metal的一部分.

压缩法线贴图(在第6.7.2节中讨论)需要格外小心.专为RGB颜色设计的压缩格式通常无法正常显示数据.大多数方法都利用这样的事实,即已知法线为单位长度,并进一步假定其分量为正(正切空间法线的合理假设).这仅允许存储法线的和分量.分量是动态导出的

由于仅存储两个分量,而不是三个,因此它本身会导致适度的压缩.由于大多数GPU本身都不支持三组件纹理,因此这也避免了浪费组件的可能性(或必须在第四组件中包装其他数量的组件).通常将和分量存储在BC5/3Dc格式的纹理中来实现进一步的压缩.参见图6.22.由于每个块的参考值划分了最小和最大x和y分量值,因此可以将它们视为在xy平面上定义边界框.三位插值因子允许在每个轴上选择八个值,因此将边界框划分为8×8的可能法线网格.或者,可以使用EAC的两个通道（用于x和y）,然后按照上面的定义计算z.

在不支持BC5/3Dc或EAC格式的硬件上,常见的后退[1227]是使用DXT5格式的纹理并将两个组件存储在绿色和alpha组件中(因为这些组件的存储精度最高).其他两个组件未使用.

PVRTC[465]是Imagination Technologies称为PowerVR的硬件上可用的纹理压缩格式,其最广泛的用途是用于iPhone和iPad.它为每个纹理像素提供2位和4位的方案,并压缩4×4纹理像素的块.关键思想是提供图像的两个低频(平滑)信号,这些信号是使用相邻的像素数据块和插值获得的.然后,每个纹理像素使用1或2位在图像上的两个信号之间进行插值.

自适应可伸缩纹理压缩（ASTC）[1302]的不同之处在于它,将n×m纹理像素的块压缩为128位.块大小从4×4到12×12不等,这导致了不同的比特率,从每texel低至0.89位到每texel高达8位.ASTC使用各种各样的技巧来实现紧凑的索引表示,并且可以为每个块选择行数和端点编码.另外,ASTC可以处理每个纹理1至4个通道以及LDR和HDR纹理.ASTC是OpenGL ES 3.2及更高版本的一部分.

上面介绍的所有纹理压缩方案都是有损的,并且在压缩纹理时,可以在此过程上花费不同的时间.花几秒钟甚至几分钟压缩,可以获得更高的质量.因此,这通常是作为脱机预处理完成的,并存储起来供以后使用.或者,一个人只能花费几毫秒的时间,结果质量较低,但是纹理可以实时压缩并立即使用.一个例子是天空盒(第13.3节),每隔一秒左右就会重新生成一次,此时云层可能已经略微移动了.解压过程非常快,因为它是使用固定功能的硬件完成的.这种差异称为数据压缩不对称,其中压缩可以而且确实比解压缩花费更长的时间.

Kaplanyan[856]提出了几种可以改善压缩纹理质量的方法.对于包含颜色的纹理和法线贴图,建议使用每个组件16位的方式创作贴图.对于彩色纹理,然后执行直方图重归一化(在这16位上),然后使用着色器中的比例和偏置常数(每个纹理)反转其效果.直方图归一化是一种将图像中使用的值扩展到整个范围的技术,这实际上是一种对比度增强.每个分量使用16位可确保在重新归一化后直方图中没有闲置的时隙,从而减少了许多纹理压缩方案可能引入的条带失真.如图6.23所示.此外,如果75％的像素高于116/255，Kaplanyan建议对纹理使用线性颜色空间,否则将纹理存储在sRGB中.对于法线贴图,他还指出BC5/3Dc通常独立于y来压缩x,这意味着并非总能找到最佳法线.相反,他建议对法线使用以下错误度量:

其中是原始法线,并且是相同的法线压缩,然后解压缩.

应当注意,还可以在不同的颜色空间中压缩纹理,这可以用来加速纹理压缩.常用的变换是[**1112**]:

其中,是亮度项,和是色度项.逆变换也很廉价:

这相当于增加了一些.这两个变换是线性的,这可以从公式6.6中看出,它是矩阵矢量乘法,其本身是线性的(请参见公式4.1和4.2).这很重要,因为可以存储,而不是将RGB存储在纹理中.纹理硬件仍可以在空间中执行过滤,然后像素着色器可以根据需要转换回RGB.应该注意的是,这种变换本身是有损的,这可能或可能不重要.

还有另一个可逆的变换,总结为

其中表示右移操作符.这意味着可以在例如24位RGB颜色和相应的表示之间进行来回转换而不会造成任何损失.应该注意的是,如果RGB中的每个分量都具有位,则和均具有位,以确保可逆变换.虽然只需要位.Van Waveren和Castano[**1852**]使用有损变换在CPU或GPU上实现对DXT5/BC3的快速压缩.它们将Y存储在alpha通道中(因为它具有最高的精度）,而和存储在RGB的前两个分量中.由于分别存储和压缩,因此压缩变得很快.对于和组分,他们找到一个二维边界框,并选择产生最佳结果的对角线框.请注意,对于在CPU上动态创建的纹理,最好也压缩CPU上的纹理.通过在GPU上渲染来创建纹理时,通常最好也压缩GPU上的纹理.YCoCg变换和其他亮度色度变换通常用于图像压缩,其中色度分量在2×2像素上平均.这样可以减少50％的存储量,并且通常可以正常工作,因为色度趋于缓慢变化.Lee-Steere和Harmon [**1015**]通过将其转换为色相饱和度值(HSV),对色相和饱和度下采样x和y的系数为4,并将值存储为单通道DXT1纹理,进一步迈出了一步.Van Waveren和Castano也描述了法线图压缩的快速方法[**1853**].

Griffin和Olano[**601**]的一项研究表明,将多个纹理应用于具有复杂阴影模型的几何模型时,纹理的质量通常很低,而没有任何明显的差异.因此,根据使用情况,降低质量是可以接受的. Fauconneau[**463**]提出了DirectX 11纹理压缩格式的SIMD实现.

**6.3 生成纹理** 2020年12月4日11点28分

给定纹理空间位置,执行图像查找是生成纹理值的一种方法.另一个是评估函数,从而定义**过程纹理**.

尽管过程纹理通常在脱机渲染应用程序中使用,但图像纹理在实时渲染中更为常见.这是由于现代GPU中图像纹理化硬件的极高效率,它可以在一秒钟内执行数十亿次纹理访问.但是,GPU体系结构正在朝着更便宜的计算和(相对)更昂贵的内存访问发展.这些趋势已使过程纹理在实时应用程序中得到更多使用.

鉴于体积图像纹理的高存储成本,体积纹理对于过程纹理特别有吸引力.可以通过多种技术来合成这样的纹理.最常见的一种方法是使用一个或多个噪声函数来生成值[**407,1370,1371, 1372**].参见图6.24.噪声函数通常在两个连续的二次方频率(八度)上采样.每个八度都有权重,通常随着频率的增加而降低,这些加权样本的总和称为**湍流**函数.

由于评估噪声函数的成本，通常会预先计算三维阵列中的晶格点，并将其用于内插纹理值.有多种使用颜色缓冲区混合来快速生成这些数组的方法[1192].Perlin[1373]提出了一种快速,实用的方法来对该噪声函数进行采样,并展示了一些用途. Olano[1319]提供了噪声生成算法,允许在存储纹理和执行计算之间进行权衡。 McEwan等人[1168]开发了无需任何查找即可在着色器中计算经典噪声和单纯形噪声的方法,并且提供了源代码. Parberry [1353]使用动态编程在多个像素上摊销计算,以加快噪声计算.Green [587]提供了一种更高质量的方法,但是对于接近交互的应用程序,它意味着更多的方法,因为它使用50像素着色器指令进行一次查找.Perlin [1370，1371，1372]提出的原始噪声函数可以得到改善。 Cook和DeRose [290]提出了另一种表示形式，称为小波噪声，它避免了混叠问题，而评估成本仅增加了一点点。刘等人[1054]使用各种噪声函数来模拟不同的木材纹理和表面光洁度。我们也推荐Lagae等人[956]关于该主题的最新报告.

其它程序方法也是可能的.例如,通过测量从每个位置到散布在空间中的一组“特征点”的距离来形成细胞纹理.以各种方式(例如,更改颜色或着色法线)映射所得的最接近距离,可以创建看起来像细胞,石板,蜥蜴皮和其他自然纹理的图案. Griffiths[**602**]讨论了如何有效地找到最接近的邻居并在GPU上生成蜂窝纹理.

程序纹理的另一种类型是物理模拟或其他一些交互过程的结果，例如水波纹或扩展裂缝.在这种情况下,过程纹理可以对动态条件做出有效的无限变化.

当生成程序化二维纹理时,参数化问题可能比创作的纹理面临更大的困难,在创作的纹理中,可以手动修改或解决拉伸或接缝伪像.一种解决方案是通过将纹理直接合成到表面上来完全避免参数化.在复杂的表面上执行此操作在技术上具有挑战性,并且是研究的活跃领域.有关该领域的概述,请参见Wei等人[**1861**].

抗锯齿的程序纹理比抗锯齿的图像纹理既困难又容易.一方面,诸如mipmapping之类的预计算方法不可用,这给程序员带来了负担.另一方面,过程纹理作者拥有有关纹理内容的“内部信息”，因此可以对其进行定制以避免混叠.对于通过将多个噪声函数求和而创建的过程纹理尤其如此.每个噪声函数的频率都是已知的,因此可以丢弃任何会导致混叠的频率,从而实际上降低了计算成本.有多种技术可以消除其他类型的过程纹理的锯齿[407,605,1392,1512].Dorn等[371]讨论了以前的工作,并提出了一些重新构造纹理函数以避免高频(即频带受限)的方法.

**6.4 纹理动画** 2020年11月26日19点18分

应用于表面的图像不必是静态的.例如,视频源可以用作随帧变化的纹理.

纹理坐标也不必是静态的.应用程序设计人员可以在网格的数据本身中或通过应用在顶点或像素着色器中的函数,在帧之间显式更改纹理坐标.想象一下,已经对一个瀑布进行了建模,并用看起来像落水的图像对其进行了纹理处理.说v坐标是流向.为了使水运动,必须从每个连续帧的v坐标中减去一个量.从纹理坐标中减去的效果是使纹理本身看起来向前移动.

通过将矩阵应用于纹理坐标可以创建更精细的效果.除了平移之外,这还允许进行线性变换,例如缩放,旋转和剪切[1192,1904],图像扭曲和变形变换[1729]和广义投影[638].通过在CPU或着色器中应用功能,可以创建更多精致的效果.

通过使用纹理混合技术,可以实现其他动画效果.例如,从大理石纹理开始并以肉色褪色,可以使雕像栩栩如生[1215].

**6.5 材质映射** 2020年11月26日19点24分

纹理的常见用途是修改影响着色方程的材料属性.现实世界中的对象通常具有在其表面上变化的材质属性.为了模拟此类对象,像素着色器可以从纹理读取值,并在评估着色方程之前使用它们修改材质参数.最常被纹理修改的参数是表面颜色.这种纹理称为反照率颜色贴图或漫反射颜色图.但是,可以通过纹理修改任何参数:替换,乘以或以其他方式更改它.例如,在图6.25中,将三个不同的纹理应用于表面,以替换常量值.

在材料中纹理的使用可以更进一步.代替修改方程式中的参数,可以使用纹理来控制像素着色器本身的流和功能.通过使一种纹理指定表面的哪些区域具有哪种材质,可以将具有不同着色方程式和参数的两种或多种材质应用于表面,从而为每种材质执行不同的代码.例如,具有一些生锈区域的金属表面可以使用纹理来指示生锈的位置,根据该纹理查找有条件地执行着色器的生锈部分,否则执行闪亮的金属着色器(第9.5.2节).

着色模型输入(例如表面颜色)与从着色器输出的最终颜色具有线性关系。因此，可以使用标准技术过滤包含此类输入的纹理,并避免混叠.包含非线性阴影输入(例如粗糙度或凹凸贴图)的纹理（第6.7节）需要多加注意，以避免混叠。考虑到着色方程的滤波技术可以改善此类纹理的结果。这些技术在9.13节中讨论.

**6.6 Alpha映射** 2020年11月26日19点25分

使用alpha混合或alpha测试可以将alpha值用于许多效果,例如有效地渲染树叶,爆炸和远处的物体,仅举几例.本部分讨论了将纹理与Alpha配合使用的方法,并指出了各种限制和解决方案.

一种与纹理相关的效果是贴花.例如,假设您想在茶壶上放一朵花的图片.您不想要整个图片,而只想要花所在的部分.通过将0的alpha值分配给texel,可以使其透明,从而使其无效.因此,通过正确设置贴花贴图的Alpha,你可以用贴花替换或混合基础表面.通常,夹具对应功能与透明边框一起使用，以将贴花的单个副本(相对于重复纹理)应用到表面。图6.26中显示了如何执行贴花的示例.有关贴花的更多信息,请参见第20.2节.

alpha的类似应用是在制作抠图.假设您制作了灌木的贴花图像,并将其应用于场景中的矩形.其原理与贴花的原理相同,除了将衬套绘制在其下面任何几何形状的顶部之外,而不是与下层表面齐平.这样,使用单个矩形可以渲染具有复杂轮廓的对象.

在衬套的情况下,如果围绕它旋转查看器,则幻觉会失败,因为衬套没有厚度.一种解决方法是复制此灌木矩形并将其沿树干旋转90度.这两个矩形构成了廉价的三维灌木丛,有时也称为“十字树”[**1204**],从地面上看时,这种错觉相当有效.参见图6.27. Pelzer[**1367**]讨论了使用三个切口代表草的类似配置.在第13.6节中,我们讨论了一种称为广告牌的方法,该方法用于将这种渲染减少为单个矩形.如果观看者移至地面上方,则错觉就会消失,因为从上方看到灌木丛是两个切口.参见图6.28.为了解决这个问题,可以以不同的方式(切片,分支,图层)添加更多的抠图,以提供更具说服力的模型.第13.6.5节讨论了一种生成此类模型的方法；第857页的图19.31显示了另一个.有关最终结果的示例,请参见第2和1049页上的图像.

将alpha贴图和纹理动画结合使用可以产生令人信服的特殊效果,例如火炬闪烁,植物生长,爆炸和大气效果.

有几种使用alpha贴图渲染对象的选项.Alpha混合(第5.5节)允许使用分数透明度值,这可以对对象边缘以及部分透明的对象进行抗锯齿.但是,alpha混合需要在不透明的三角形之后和从后到前的顺序渲染混合的三角形.一个简单的交叉树是两个剪切纹理的示例,其中没有正确的渲染顺序,因为每个四边形都位于另一个的前面.即使在理论上可以排序并获得正确的顺序,通常这样做效率也不高.例如,一块田地可能有成千上万个以切口为代表的草叶.每个网格物体可以由许多单独的叶片制成.明确分类每个刀片是不切实际的.

渲染时,可以通过几种不同的方法来改善此问题.一种是使用Alpha测试,这是有条件地丢弃像素着色器中具有低于给定阈值的Alpha值的片段的过程.这样做是



其中texture.a是来自纹理查找的alpha值,参数alphaThreshold是用户提供的阈值,该阈值确定哪些片段将被丢弃.该二进制可见性测试使三角形可以以任何顺序呈现,因为透明片段被丢弃了.我们通常希望对alpha为0.0的任何片段执行此操作.丢弃完全透明的片段还有一个额外的好处,即可以节省进一步的着色器处理和合并成本,还可以避免将z缓冲区中的像素错误地标记为可见[**394**].对于切口,我们通常将阈值设置为高于0.0,例如0.5或更高,然后采取进一步的步骤,然后完全忽略alpha值,而不是将其用于混合.这样做可以避免乱码.但是,由于只有两个级别的透明度(完全不透明和完全透明)可用,所以质量较低.另一种解决方案是对每个模型执行两次遍历-一次遍历实体切口(将其写入z缓冲区),另一遍进行半透明样本(不进行).

alpha测试还有其他两个问题,即放大倍数[**1374**]和缩小倍数[**234,557**].将alpha测试与mipmapping一起使用时,如果处理方式不同,效果可能令人难以置信.图6.29的顶部显示了一个示例,其中树的叶子变得比预期的更加透明.这可以用一个例子来解释.假设我们有一个具有四个alpha值的一维纹理,即(0.0,1.0,1.0,0.0).通过平均,下一个mipmap级别变为(0.5,0.5),然后最高级别为(0.5).现在,假设我们使用αt= 0.75.访问mipmap级别0时,可以显示出4中的1.5纹素将通过丢弃测试.但是,访问下两个级别时,由于0.5 <0.75，所有内容将被丢弃.请参见图6.30.

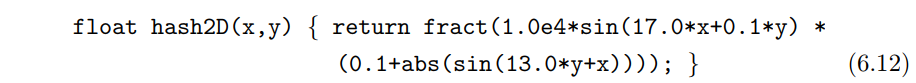
Castano[**234**]提出了一种在mipmap创建期间完成的简单解决方案,效果很好.对于mipmap级别,覆盖率定义为

其中是在mipmap级别中的纹素数量,是在像素处来自mipmap级别的alpha值,而是用户在公式6.9中提供的alpha阈值.在这里,我们假设的结果为1,否则为0.注意,表示最低的mipmap级别,即原始图像.然后,对于每个mipmap级别,我们找到一个新的mipmap阈值,而不是使用,从而使等于(或尽可能接近).可以使用二进制搜索来完成.最后,在mipmap级别中所有纹理像素的alpha值按缩放.图6.29的底部使用了此方法,NVIDIA的纹理工具对此方法提供了支持. Golus[**557**]给出了一个变体,其中未修改mipmap,但是随着mipmap级别的增加,alpha会在着色器中按比例放大.

Wyman和McGuire[**1933**]提出了另一种解决方案,其中理论上将等式6.9中的代码行替换为



随机函数在[0,1]中返回一个均匀值,这意味着平均而言,它将得出正确的结果.例如,如果纹理查找的alpha值为0.3,则片段将以30％的机会被丢弃.这是一种随机透明的形式,每个像素只有一个样本[**423**].在实践中,将随机函数替换为散列函数以避免时空高频噪声:



通过对上述函数的嵌套调用来形成三维哈希,即float hash3D（x，y，z）{return hash2D（hash2D（x，y），z）; },返回[0，1）之间的数字.哈希的输入是对象空间坐标除以对象空间坐标的最大屏幕空间导数(x和y），然后进行钳位.需要进一步注意以获得z方向运动的稳定性,并且该方法最好与时间抗锯齿技术结合使用.该技术随着距离的增加而逐渐淡入,因此在关闭时我们根本不会获得任何随机效果.这种方法的优点是每个片段平均而言都是正确的,而Castano的方法[**234**]为每个mipmap级别创建单个.但是,此值可能会在每个mipmap级别上有所不同,这可能会降低质量并需要艺术家干预.

Alpha测试在放大下显示波纹伪影,可以通过将Almap预先计算为距离场[**580**]来避免（请参见第677页的讨论).

对覆盖的Alpha和类似的功能透明度自适应抗锯齿,采用片段的透明度值,并将其转换为像素内覆盖了多少个样本[**1250**].这个想法就像5.5节中描述的屏幕门透明度,但在亚像素级别.想象每个像素有四个样本位置,并且一个片段覆盖一个像素,但是由于有切口纹理,因此它是25％透明(75％不透明)的.Alpha至覆盖模式使片段变得完全不透明,但仅覆盖了四个样本中的三个.此模式对于重叠草叶的抠图纹理很有用,例如[**887,1876**].由于每个绘制的样本都是完全不透明的，因此最接近的叶状体将沿其边缘以一致的方式将对象隐藏在其后方。由于Alpha混合功能已关闭，因此无需排序即可正确混合半透明边缘像素.

Alpha to Coverage是抗锯齿Alpha测试的好方法,但是在Alpha混合时会显示伪影.例如,两个具有相同alpha覆盖率的alpha混合片段将使用相同的子像素图案,这意味着一个片段将完全覆盖另一个片段,而不是与其混合. Golus[**557**]讨论了使用fwidth()着色器指令为内容提供更清晰的边缘.参见图6.31.

对于alpha映射的任何使用,重要的是要了解双线性插值如何影响颜色值.想象一下彼此相邻的两个纹理像素:rgbα=（255，0，0，255）是纯红色,而它的邻居rgbα=（0，0，0，2）是黑色的,几乎完全透明.正好位于两个纹理像素中间的位置的rgbα是多少？简单插值得出（127，0，0，128）,仅所得的rgb值就有一个“较暗”的红色.但是,此结果实际上并不是暗淡的,它是一个全红色,已预先乘以其alpha.如果您对Alpha值进行插值,则为确保正确的插值,您需要确保在插值之前,要插值的颜色已经预先乘以Alpha.例如,假设将几乎透明的邻居设置为rgbα=（0，255，0，2）,并给出绿色的微小色调.该颜色不会与alpha预乘,在插值时会得到结果（127，127，0，128）-绿色的微小调子突然将结果变为（预乘）黄色样本.此邻居纹理元素的预乘版本为（0，2，0，2）,这给出了正确的预乘结果（127，1，0，128）。这个结果更有意义,最终的预乘颜色主要是红色,而绿色则难以察觉.

忽略双线性插值的结果给出了预乘结果,可能会导致贴花和剪切对象周围出现黑色边缘.“较暗”的红色结果被其余管道视为未相乘的颜色,并且条纹变为黑色.即使使用alpha测试,此效果也可以看到.最好的策略是在完成双线性插值之前进行预乘[**490，648，1166，1813**]. WebGL API支持此功能,因为合成对于网页很重要.但是,双线性插值通常由GPU执行,并且在执行此操作之前,着色器无法完成对纹理像素值的操作.图像不会以PNG等文件格式进行预乘,因为这样做会失去色彩精度.使用alpha映射时,默认情况下,这两个因素共同导致黑边现象.一种常见的解决方法是对剪切图像进行预处理,用从附近的不透明纹理像素派生的颜色绘制透明的“黑色”纹理像素[**490,685**].通常,所有透明区域都需要以这种方式手动或自动重新粉刷,以使mipmap级别也避免出现边缘问题[**295**].还值得注意的是,在形成具有Alpha值的Mipmap时应使用预乘值[**1933**].

**6.7 Bump Mapping** 2020年11月26日20点09分

本节描述了一大类小型细节表示技术,我们统称为**凹凸贴图**.所有这些方法通常都是通过修改每个像素的着色例程来实现的.与单独的纹理贴图相比,它们具有更三维的外观,但没有添加任何其他几何形状.

一个物体的细节可以分为三个等级:覆盖许多像素的宏观特征,横跨几个像素的细观特征和实质上小于一个像素的微观特征.这些类别有些不稳定,因为在动画或交互式会话期间,观看者可能在许多距离处观察到同一对象.

宏观几何由顶点和三角形或其他几何图元表示.创建三维角色时,通常以宏观尺度对肢体和头部进行建模.微几何封装在着色模型中,该模型通常在像素着色器中实现,并使用纹理贴图作为参数.所用的着色模型可模拟表面微观几何形状的相互作用,例如,发光物体在微观上是光滑的,而漫射表面在微观上是粗糙的.角色的皮肤和衣服似乎具有不同的材质,因为它们使用不同的着色器,或者至少使用这些着色器中的不同参数.

中观几何描述了这两个尺度之间的一切.它包含的细节过于复杂,无法使用单个三角形进行有效渲染,但是对于观察者来说,它足以区分表面曲率在几个像素上的变化.角色脸上的皱纹,肌肉结构细节以及衣服上的褶皱和接缝都是中尺度的.中尺度建模通常使用一类统称为凹凸贴图技术的方法.这些以像素级别调整阴影参数的方式,使观看者感觉到远离基本几何形状的微小干扰,而基本几何形状实际上保持平坦.不同种类的凹凸贴图之间的主要区别是它们如何表示细节特征.变量包括现实水平和细节特征的复杂性.例如,数字艺术家通常将细节雕刻到模型中,然后使用软件将这些几何元素转换为一种或多种纹理,例如凹凸纹理和缝隙变暗的纹理.

Blinn在1978年提出了在纹理中编码中尺度细节的想法[**160**].他观察到,如果在着色过程中用稍微扰动的表面法线代替真实的表面,则表面似乎具有小范围的细节.他将描述微扰到表面法线的数据存储在数组中.

关键思想是,我们不使用纹理来更改照明方程中的颜色分量,而是访问纹理来修改表面法线.表面的几何法线保持不变.我们仅修改照明方程式中使用的法线.此操作没有物理等效项.我们在曲面法线上执行更改,但是曲面本身在几何意义上保持平滑.正如具有每个顶点的法线会给人一种幻觉,即三角形之间的表面是光滑的一样,修改每个像素的法线会更改三角形表面本身的感知,而不会更改其几何形状.

对于凹凸贴图,法线必须相对于某个参考系改变方向.为此,将切线框架(也称为切线空间基础)存储在每个顶点上.此参考系用于将灯光转换到曲面位置的空间(反之亦然),以计算干扰法线的效果.除了顶点法线外,在多边形表面上应用了法线贴图的情况下,我们还存储了切线和双切线向量.双切线向量也被错误地称为双法线向量[**1025**].

切线和切线向量表示法线贴图本身在对象空间中的轴,因为目标是将光线变换为相对于贴图.参见图6.32.

法线,切线和双切线这三个向量形成基本矩阵:

这个矩阵有时缩写为TBN,它将光的方向(对于给定的顶点)从世界空间转换为切线空间.这些向量不必彼此真正垂直,因为法线贴图本身可能会变形以适合曲面.但是,非正交的基会导致纹理倾斜,这可能意味着需要更多的存储空间,并且可能会对性能产生影响,即矩阵无法通过简单的转置来反转[**494**].一种节省内存的方法是只在顶点存储切线和双切线,并取它们的叉积来计算法线.但是,只有在矩阵的手性始终相同的情况下,此技术才有效[**1226**].通常,模型是对称的:飞机,人,文件柜和许多其它物体.由于纹理消耗大量内存,因此它们通常被镜像到对称模型上.因此,仅存储对象纹理的一侧,但是纹理映射将其放置在模型的两侧.在这种情况下,切线空间的旋向性在两侧会有所不同,因此无法假定.如果在每个顶点存储了额外的信息位来指示惯用性,那么在这种情况下仍然可以避免存储法线.如果已设置,则此位用于求反切线和双切线的叉积,以生成正确的法线.如果切线框架是正交的,则还可以将基础存储为四元数(第4.3节),这既节省空间,又可以节省每个像素的一些计算[**494,1114,1154,1381,1639**].尽管在实践中很少见,但质量可能会略有下降.

切线空间的概念对于其他算法很重要.如下一章所述,许多着色方程仅依赖于曲面的法线方向.但是,与表面相比,拉丝铝或天鹅绒等材料也需要知道观察者和照明的相对方向.切线框架可用于定义材料在表面上的方向. Lengyel[**1025**]和Mittring[**1226**]的文章提供了该领域的广泛报道.舒勒[**1584**]提出了一种在像素着色器中动态计算切线空间的方法,而无需为每个顶点存储预先计算的切线帧. Mikkelsen[**1209**]对这一技术进行了改进,并推导了一种不需要任何参数化方法,而是使用表面位置的导数和高度场的导数来计算扰动法线的方法.但是,与使用标准切线空间映射相比,此类技术可能导致显示的细节明显减少,并且可能会产生美术工作流问题[**1639**].

**6.7.1 Blinn方法**

Blin最初的凹凸贴图方法在纹理的每个纹理像素上存储两个有符号的值和.这两个值对应于沿和图像轴改变法线的量.也就是说,这些纹理值(通常是双线性插值的)用于缩放垂直于法线的两个矢量.将这两个向量添加到法线以更改其方向.和这两个值描述了曲面在该点面向哪个方向.见图6.33.这种类型的凹凸贴图纹理称为偏移矢量凹凸贴图或偏移贴图.

表示凹凸的另一种方法是使用高度场来修改表面法线的方向.每个单色纹理值代表一个高度,因此在纹理中,白色是高区域,黑色是低区域(反之亦然).有关示例,请参见图6.34.这是第一次创建或扫描凹凸贴图时常用的格式,它也是1978年由Blinn引入的.高度场用于导出和符号值,类似于第一种方法中使用的值.这是通过获取相邻列之间的差异来获得的斜率以及相邻行之间的差异来获取的[**1567**].一种变体是使用Sobel滤波器,该滤波器赋予直接相邻的邻居更大的权重[**535**].

**6.7.2 法线映射** 2020年11月27日16点24分

凹凸贴图的常见方法是直接存储法线贴图.算法和结果在数学上与布林的方法相同;只有存储格式和像素着色器计算会更改.

法线贴图对映射到的进行编码,例如,对于8位纹理,轴值0表示-1.0,255表示1.0.如图6.35所示.颜色(浅蓝色)将代表所示颜色映射的平坦表面,即法线.

法线映射表示最初是作为世界空间法线映射[**274,891**]引入的,在实践中很少使用.对于这种类型的贴图,其扰动非常简单:在每个像素处,从贴图中检索法线,并将其与光的方向一起直接使用,以计算表面上该位置的着色.还可以在对象空间中定义法线贴图,以便可以旋转模型,然后法线仍然有效.但是，世界和对象空间表示都将纹理绑定到特定方向的特定几何体，这限制了纹理的重复使用.

取而代之的是,通常在切线空间(即相对于曲面本身)中检索扰动的法线.这允许表面变形以及正常纹理的最大重复使用.切线空间法线贴图也可以很好地压缩,因为分量的符号(与不受扰动的表面法线对齐的符号)通常可以假定为正.

可以使用法线贴图来提高逼真度,见图6.36.

与过滤颜色纹理相比,过滤法线贴图是一个难题.通常,法线和着色颜色之间的关系不是线性的,因此标准过滤方法可能会导致令人反感的混叠.想象一下看楼梯由闪亮的白色大理石块制成.在某些角度下,楼梯的顶部或侧面可以捕捉光线并反射出明亮的镜面高光.但是,楼梯的平均法线为45度角.它会从与原始楼梯完全不同的方向捕获亮点.如果在没有正确过滤的情况下渲染具有尖锐镜面高光的凹凸贴图,则由于高光由于样本掉落的位置而进出眨眼,会产生分散注意力的闪光效果.

朗伯曲面是一种特殊情况,其中法线贴图对着色几乎具有线性影响.朗伯着色几乎完全是一个点积,它是线性运算.平均一组法线并对其结果进行点积运算等于将单个点积与法线平均:

请注意,平均向量在使用前未标准化.公式6.14表明,对于Lambertian曲面,标准过滤和mipmap几乎可以产生正确的结果.由于Lambertian着色方程不是点积,因此结果不太正确.它是一个钳位点积-.钳位操作使其变为非线性.这将使表面过于暗淡,以至于无法直视光的方向,但是实际上这通常是不令人讨厌的[**891**].一个警告是,一些通常用于法线贴图的纹理压缩方法(例如从其他两个法线重构z分量)不支持非单位长度的法线,因此使用非标准化的法线贴图可能会带来压缩困难.

在非朗伯曲面的情况下,可以通过将着色方程的输入作为一组过滤,而不是单独过滤法线贴图来产生更好的结果.这样做的技术在9.13节中讨论.

最后,从高度图导出法线图可能很有用.如下进行[**405**].首先,使用中心差来计算x和y方向上的导数近似值

然后在texel处的非归一法线为

必须注意纹理的边界.

通过使凹凸能够将阴影投射到它们自己的表面上,可以使用水平映射[Horizon mapping][**1027**]进一步增强法线贴图.通过预先计算其他纹理(每个纹理与沿表面平面的方向相关联)并为每个纹理像素存储该方向上的地平线角度,可以完成此操作.有关更多信息,请参见第11.4节.

**6.8 透视映射** 2020年11月27日16点36分

凹凸和法线贴图的问题在于,凹凸永远不会随视角移动位置,也不会相互阻挡.例如,如果您以真实的角度看一眼真正的砖墙,您将看不到砖之间的砂浆.墙壁的凹凸贴图永远不会显示这种类型的遮挡,因为它只会改变法线.最好使凸块实际影响在每个像素处渲染表面上的哪个位置.

视差映射的概念由Kaneko[**851**]于2001年提出,并由Welsh[**1866**]进行了完善和推广.视差指的是当观察者移动时,对象的位置相对于彼此移动.当观察者移动时,凹凸应该看起来具有高度.视差映射的关键思想是通过检查发现的可见物体的高度来对像素中应该看到的物体进行有根据的猜测.

对于视差贴图,将凹凸存储在高度场纹理中.在给定像素处查看表面时,将在该位置检索高度场值,并将其用于移动纹理坐标以检索表面的不同部分.偏移量基于所获取的高度和眼睛与表面的角度.见图6.37.高度场值要么存储在单独的纹理中,要么打包为其他纹理的未使用的颜色或Alpha通道(将不相关的纹理打包在一起时必须格外小心,因为这会对压缩质量产生负面影响).在用于移动坐标之前,将对高度场值进行缩放和偏置.比例尺确定了高度场要在表面上方或下方延伸的高度,并且偏差提供了不发生偏移的“海平面”高度.给定纹理坐标位置,调整后的高度场高度和具有高度值和水平分量的归一化视图矢量,新的视差调整后的纹理坐标为

请注意,与大多数着色方程式不同,此处执行计算的空间很重要-视图矢量必须在切线空间中.

尽管是一个简单的近似值，但如果凸块高度的变化相对较慢[**1171**]，则这种移位在实践中效果很好.这样,附近的相邻纹理像素的高度大约相同,因此使用原始位置的高度作为新位置高度的估计的想法是合理的.但是,此方法在较浅的视角下会破裂.当视图矢量接近表面的水平线时,高度变化较小会导致纹理坐标偏移较大.近似失败,因为检索到的新位置与原始曲面位置几乎没有高度相关性.

为了改善这个问题,Welsh[1866]引入了偏移限制的思想.想法是将移动量限制为永远不大于获取的高度.等式是

请注意,该方程式比原始方程式计算速度更快.从几何上讲,这种解释是,高度定义了一个半径,位置不能超过该半径.如图6.38所示.

由于接近1,所以在陡峭的角度(面对面)时,该方程式与原始方程几乎相同.在较小的角度,偏移量的作用受到限制.从视觉上看,这使得在浅角度时的凹凸感减小,但这比纹理的随机采样好得多.随着视图的改变,纹理游泳也存在问题,或者对于立体渲染,观看者同时感知到两个必须给出一致的深度提示的视点[1171].即使存在这些缺点,具有偏移限制的视差映射也仅花费了一些额外的像素着色器程序指令,并且相对于基本法线映射而言,图像质量得到了显着改善. Shishkovtsov[1631]通过在凹凸贴图法线方向上移动估计位置来改善视差遮挡的阴影.

**6.8.1 透视遮挡映射** 2020年11月27日星期五16点44分

凹凸贴图不会基于高度场修改纹理坐标:它仅在某个位置改变法线着色.视差贴图提供了高度场效果的简单近似值,假设像素的高度与其相邻像素的高度大致相同.这个假设可能很快就会崩溃.颠簸也永远不会相互遮挡,也不会蒙上阴影.我们想要的是在像素处可见的东西,即视图矢量首先与高度场相交的地方.

为了以更好的方式解决此问题，一些研究人员建议沿视图矢量使用光线行进,直到找到一个(近似)交点.这项工作可以在像素着色器中完成,高度数据可以作为纹理访问.我们将对这些方法的研究归纳为视差映射技术的子集,该技术以一种或另一种方式利用射线行进 [**192,1171,1361,1424,1742,1743**].

这些类型的算法除其他名称外,称为视差遮挡映射(POM)或浮雕映射方法.关键思想是首先沿着投影矢量测试固定数量的高度场纹理样本.通常会在掠射角度为视线生成更多样本,以免错过最近的交点[**1742,1743**].检索沿射线的每个三维位置,将其转​​换为纹理空间,然后进行处理以确定其是否在高度场之上或之下.一旦找到高度场以下的样本,则其下方的数量以及上一个样本之上的数量将用于查找相交位置.见图6.39.然后使用该位置使用附加的法线贴图,颜色贴图和任何其他纹理对表面着色.多层高度场可用于产生悬垂物,独立的重叠表面以及两侧的浮雕仿冒者;请参阅第13.7节。高度场跟踪方法也可以用于使颠簸的表面在自身上投射阴影，包括硬[1171、1424]和软[1742、1743]。比较请参见图6.40.

关于这个话题有很多文献.尽管所有这些方法都沿着射线前进,但存在一些差异.可以使用简单的纹理来检索高度,但也可以使用更高级的数据结构和更高级的寻根方法.一些技术可能涉及着色器丢弃像素或写入深度缓冲区,这可能会损害性能.下面我们总结了一大堆方法,但是请记住,随着GPU的发展,最佳方法也随之发展.这种“最佳”方法取决于光线行进期间的内容和完成的步骤数.

确定两个常规样本之间的实际交点的问题是寻根问题.实际上,高度场被视为深度场,矩形的平面定义了表面的上限.这样,平面上的初始点在高度场上方.在找到高度场的上面的最后一个点和下面的第一个点之后,Tatarchuk [1742,1743]使用割线方法的单个步骤来找到一个近似解. Policarpo等人[1424]在发现的两个点之间使用二元搜索,以在更近的交叉点上进行磨练. Risser等人[1497]通过使用割线方法进行迭代来加快收敛速度.折衷方案是可以并行执行常规采样,而迭代方法需要较少的总体纹理访问,但必须等待结果并执行较慢的从属纹理获取.蛮力方法似乎在整体上表现良好[1911].

足够频繁地对高度场进行采样至关重要.McGuire和McGuire[1171]提出对mipmap查找进行偏置,并使用各向异性的mipmap以确保对高频高度场(例如代表尖峰或头发的高频高度场)进行正确采样.人们还可以以比法线贴图更高的分辨率存储高度场纹理。 最后，一些渲染系统甚至不存储法线贴图，而是倾向于使用交叉滤镜[40]从高度场中即时导出法线。 第696页的公式16.1显示了该方法.

提高性能和采样精度的另一种方法是，首先不以规则的间隔对高度场进行采样，而是尝试跳过中间的空白空间。 Donnelly [367]将高度场预处理为一组体素，在每个体素中存储它与高度场表面的距离。以这种方式，可以快速跳过中间空间，但是每个高度场的存储空间更大。 Wang等人[1844]使用五维位移映射方案来保持从各个方向和位置到表面的距离。这允许复杂的曲面，自遮蔽和其他效果，但要消耗大量内存。 Mehra和Kumar [1195]出于类似目的使用定向距离图。 Dummer [393]引入了锥步映射的概念，而Policarpo和Oliveira [1426]对其进行了改进。这里的概念是还要为每个高度场位置存储一个圆锥半径。该半径定义了射线的间隔，在该间隔中与高度场最多有一个交点。该属性允许沿射线快速跳过而不会丢失任何可能的交点，但是以需要依赖纹理读取为代价。另一个缺点是创建圆锥台阶图所需的预计算，这使得该方法无法用于动态更改高度场。 Schroders和Gulik [1581]提出了四叉树浮雕映射，这是一种在遍历期间跳过体积的分层方法。 Tevs等人[1760]使用“最大mipmap”来进行跳过，同时将预计算成本降至最低。 Drobot [377]还使用存储在mipmap中的类似四叉树的结构来加速遍历，并提出了一种在不同的高度场之间进行混合的方法，其中一种地形类型转换为另一种地形类型.

上述所有方法的一个问题是，幻觉沿着对象的轮廓边缘分解，这将显示原始表面的平滑轮廓。参见图6.41。关键思想是渲染的三角形定义了应由像素着色器程序评估哪些像素，而不是表面实际位于的位置。另外，对于弯曲表面，轮廓问题变得更加复杂。 Oliveira和Policarpo [1325，1850]描述和开发了一种方法，该方法使用二次轮廓逼近技术。 Jeschke等人[824]和Dachsbacher等人[323]都给出了一种更通用，更鲁棒的方法（并回顾了以前的工作）来正确处理轮廓和曲面。最初由Hirche [750]探索，其总体思想是向外挤出网格中的每个三角形并形成一个棱镜。渲染此棱镜会强制评估可能会出现高度场的所有像素。这种方法称为壳映射，因为扩展的网格在原始模型上形成了一个单独的壳。通过保留棱镜与光线相交时的非线性特性，可以实现高度场的无伪影渲染，尽管计算成本很高。图6.42显示了这种技术的令人印象深刻的用法。

6.9 纹理光源 2020年11月27日17点03分

纹理还可以用于为光源增加视觉丰富度，并允许复杂的强度分布或聚光灯功能。对于所有照明都限于圆锥形或截头圆锥形的灯，可以使用投影纹理来调制光强度[1192，1597，1904]。这样就可以形成聚光灯，图案光，甚至是“幻灯机”效果（图6.43）。在专业剧院和电影照明中使用的切口术语之后，这些灯通常称为图案灯或曲奇灯。有关以类似方式投射阴影的投影映射的讨论，请参见第7.2节。对于不限于平截头体但可以在所有方向照亮的灯光，可以使用立方体贴图来调制强度，而不是使用二维投影纹理。一维纹理可用于定义任意距离衰减函数。结合二维角度衰减图，可以考虑复杂的体积照明模式[353]。更普遍的可能性是使用三维（体积）纹理控制光的衰减[353、535、1192]。这允许任意数量的效果，包括光束。此技术占用大量内存（所有卷纹理也是如此）。如果光线的影响量沿三个轴对称，则可以通过将数据镜像到每个八分圆中来将内存占用空间减少八倍。可以将纹理添加到任何灯光类型，以启用其他视觉效果。带纹理的灯光使艺术家可以轻松控制照明，而艺术家可以简单地编辑所使用的纹理。