

描影法

让我们继续使我们的图像更加逼真;在本章中，我们将研究如何向场景添加光源以及如何照亮它包含的对象。首先，让我们看一些术语。

阴影与照明

本章的标题是“阴影”，而不是“照明”;这是两个不同但密切相关的概念。**照明**是指计算光线对场景中单个点的影响所需的数学和算法;**着色**涉及将光在一组离散点上的效果扩展到整个对象的技术。

在第3章(光)中，我们研究了我们需要了解的有关照明的所有信息。我们可以定义环境光、点光和定向光，我们可以计算

给定场景中任何点的照明，给定其位置和该点的表面法线：

$$I_P = I_A + \sum_{i=1}^n I_i \cdot \left[\frac{\langle \vec{N}, \vec{L}_i \rangle}{|\vec{N}| |\vec{L}_i|} + \left(\frac{\langle \vec{R}_i, \vec{V} \rangle}{|\vec{R}_i| |\vec{V}|} \right)^s \right]$$

此照明方程表示光线如何照亮场景中的点。这在我们的光线追踪器中的工作方式与在我们的光栅器中的工作方式完全相同。

我们将在本章中探讨的更有趣的部分是如何将我们开发的“点照明”算法扩展到“三角形每个点的照明”算法。

平面阴影

让我们从简单开始。由于我们可以计算一个点的照明，因此我们可以只选择三角形中的任何点(例如，它的中心)，计算该点的照明，并使用它来着色整个三角形。要进行实际着色，我们可以将三角形的颜色乘以照明值。图13-1显示了结果。

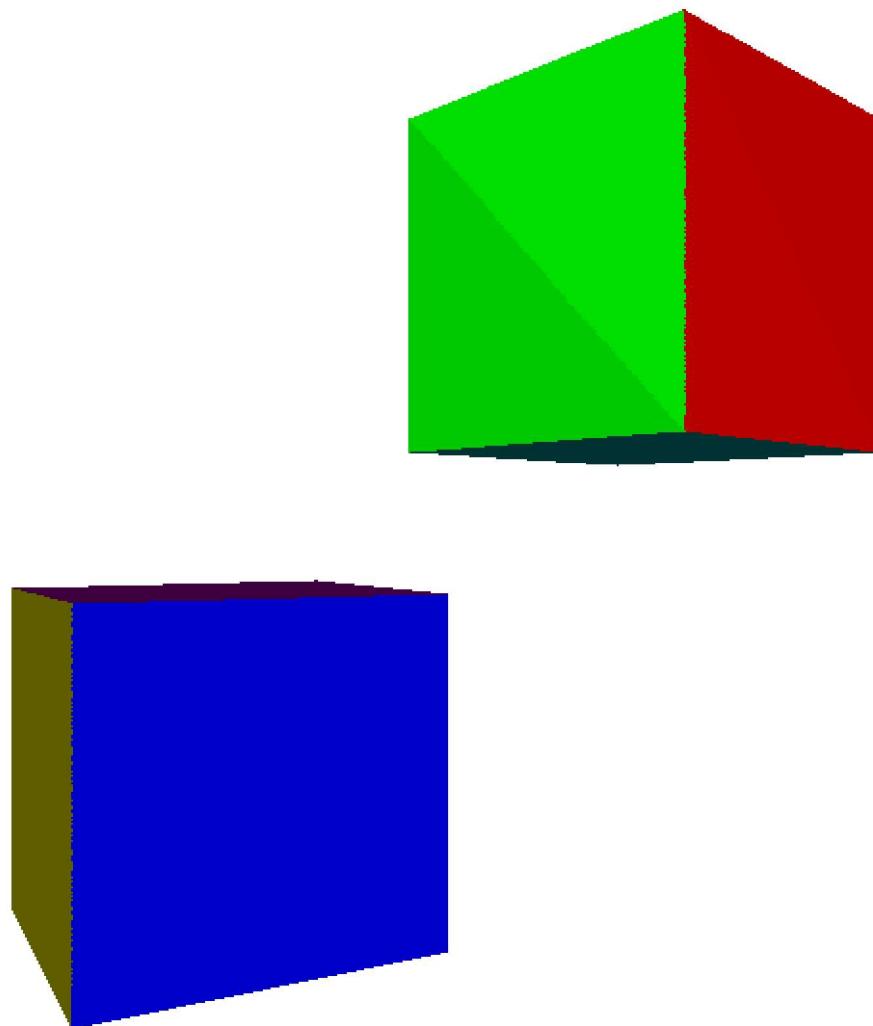


图 13-1：在平面阴影中，我们计算三角形中心的照明并将其用于整个三角形。

结果是有希望的。三角形中的每个点都具有相同的法线，因此只要光线离它相当远，每个点的光矢量就近似平行，并且每个点接收的光量大致相同。构成立方体两侧的两个三角形之间的不连续性，特别是在图 13-1 中的绿色表面上可见，是光矢量近似但不完全平行的结果。

那么，如果我们尝试使用每个点都有不同法线的对象（如图 13-2 中的球体）进行这种技术，会发生什么？

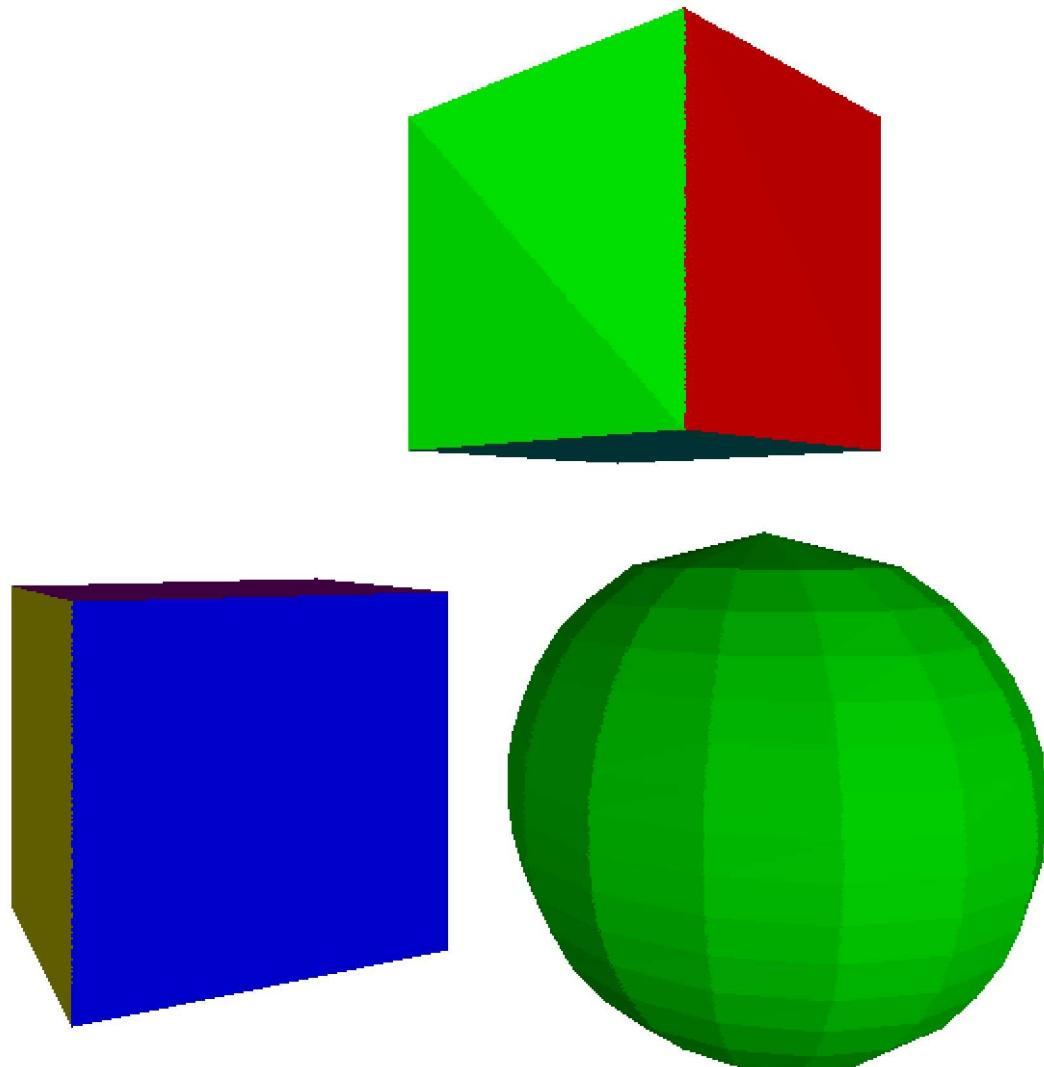


图 13-2：平面着色对于具有平面的对象效果相当好，但对于应该弯曲的对象则不是很好。

不太好。很明显，该物体不是一个真正的球体，而是由扁平的三角形斑块组成的近似值。因为这种照明使弯曲的物体看起来是平坦的，所以它被称为平面阴影。

古劳德阴影

我们如何消除照明中的这些不连续性？我们不是只计算三角形中心的照明，而是计算三角形三个顶点的照明。这为我们提供了三个照明值0.0和1.0，三角形的每个顶点对应一个。这使我们处于与[第8章（阴影三角形）](#)完全相同的情况：我们可以直接使用，使用照明值作为“强度”属性。DrawShadedTriangle

这种技术被称为*Gouraud着色*，以Henri Gouraud的名字命名，他在1971年提出了这个想法。图 13-3 显示了将其应用于立方体和球体的结果。

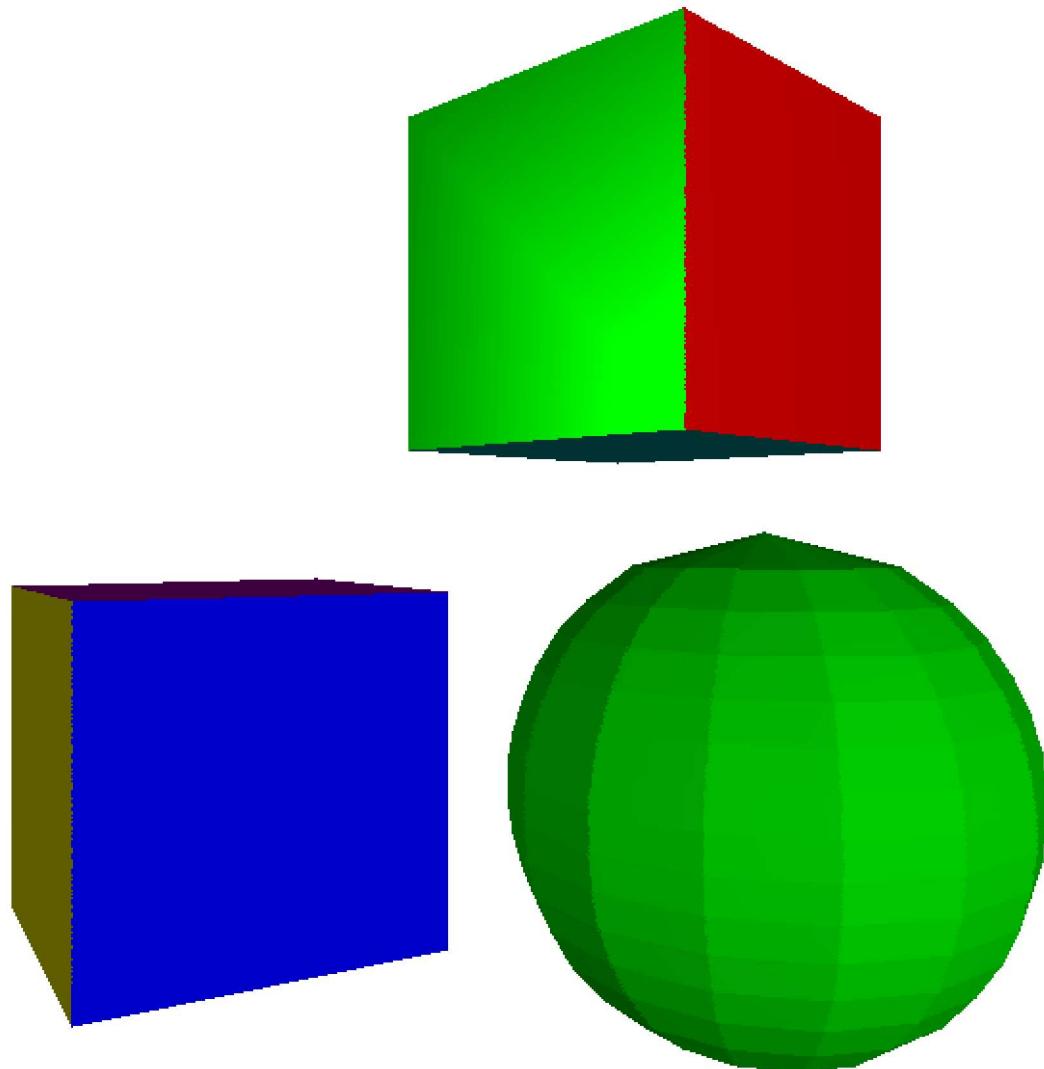


图 13-3：在 Gouraud 着色中，我们计算三角形顶点处的照明，并将它们插值到其表面上。

立方体看起来更好：不连续性消失了，因为每个面的两个三角形共享两个顶点，并且它们具有相同的法线，因此这两个三角形在这两个顶点处的照明是相同的。

然而，球体看起来仍然是多面的，其表面上的不连续性看起来真的很不对劲。这并不奇怪：我们将球体视为平面的集合。特别是，尽管每个三角形都与其相邻三角形共享顶点，但它们具有不同的法线。图 13-4 显示了该问题。

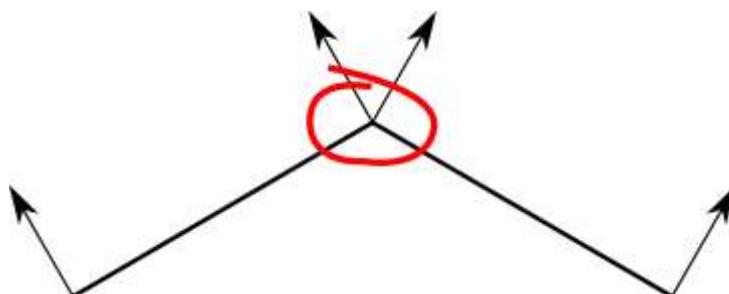


图 13-4：我们得到共享顶点处的两个不同照明值，因为它们取决于三角形的法线，而三角形是不同的。

让我们退后一步。我们使用平面三角形来表示弯曲物体的事实是我们技术的局限性，而不是物体本身的属性。

球体模型中的每个顶点对应于球体上的一个点，但它们定义的三角形只是其表面的近似值。最好使模型中的顶点尽可能接近地表示球体中的点。这意味着，除其他事项外，还要为每个顶点使用实际的球体法线，如图 13-5 所示。

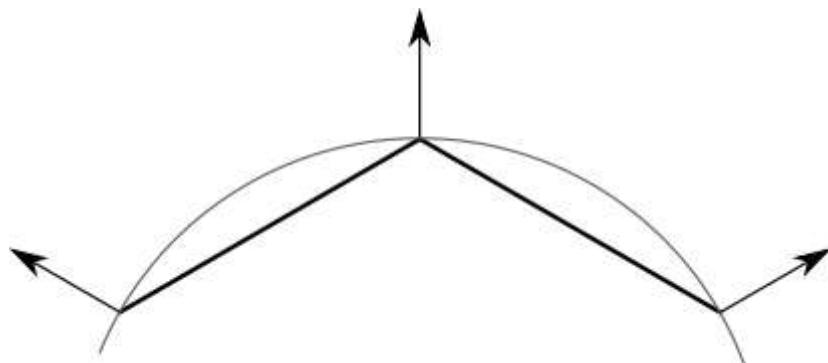


图 13-5：我们可以为每个顶点提供它所表示的曲面的法线。

请注意，这不适用于多维数据集；即使三角形共享顶点位置，每个面也需要独立于其他面进行着色。立方体的顶点没有单一的“正确”法线。

我们的渲染器无法知道模型应该是弯曲对象的近似值还是平面对象的精确表示。毕竟，立方体是球体的非常粗略的近似值！为了解决这个问题，我们将三角形法线作为模型的一部分，以便其设计者可以做出这个决定。

某些对象（如球体）的每个顶点都有一个法线。其他对象（如立方体）对于使用顶点的每个三角形具有不同的法线。所以我们不能使法线成为顶点的属性；它们必须是使用它们的三角形的属性：

```
model {
    name = cube
    vertices {
        0 = (-1, -1, -1)
        1 = (-1, -1, 1)
        2 = (-1, 1, 1)
        ...
    }
    triangles {
        0 = {
            vertices = [0, 1, 2]
            normals = [(-1, 0, 0), (-1, 0, 0), (-1, 0, 0)]
        }
        ...
    }
}
```

图 13-6 显示了使用 Gouraud 着色和适当的顶点法线渲染的场景。

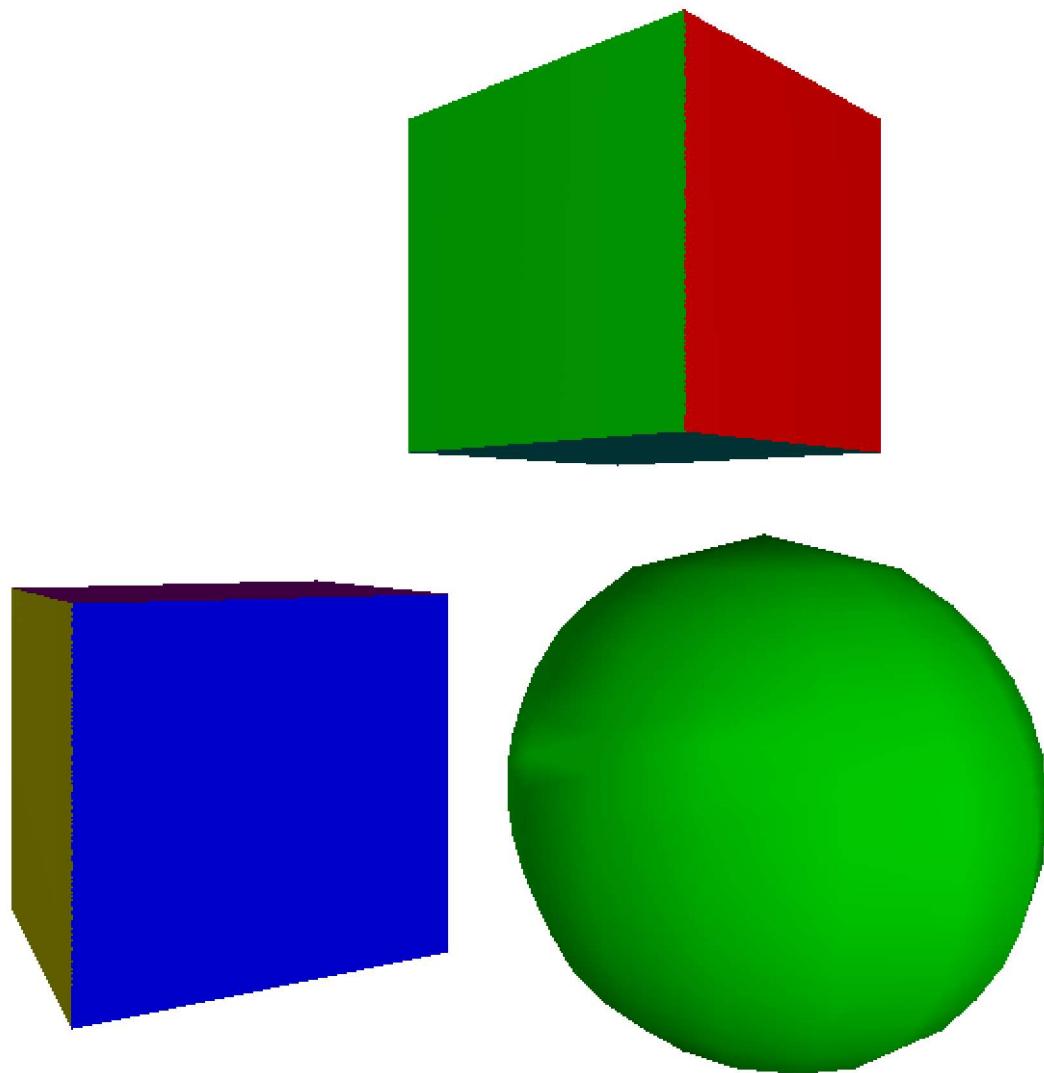


图 13-6：模型中指定的法线向量的 Gouraud 着色。立方体看起来仍然像立方体，而球体现在看起来像一个球体。

立方体看起来仍然像立方体，而球体现在看起来非常像一个球体。事实上，你只能通过看它的轮廓来判断它是由三角形组成的。这可以通过使用更多、更小的三角形来改进，但代价是需要更多的计算能力。

不过，当我们尝试渲染闪亮的物体时，Gouraud 着色开始崩溃；球体上的镜面反射高光显然是不现实的。

这表明存在更普遍的问题。当我们把点光源移动到非常靠近大脸的位置时，我们自然会期望它看起来更亮，镜面反射效果会变得更加明显；然而，Gouraud 着色会产生完全相反的结果（图 13-7）。

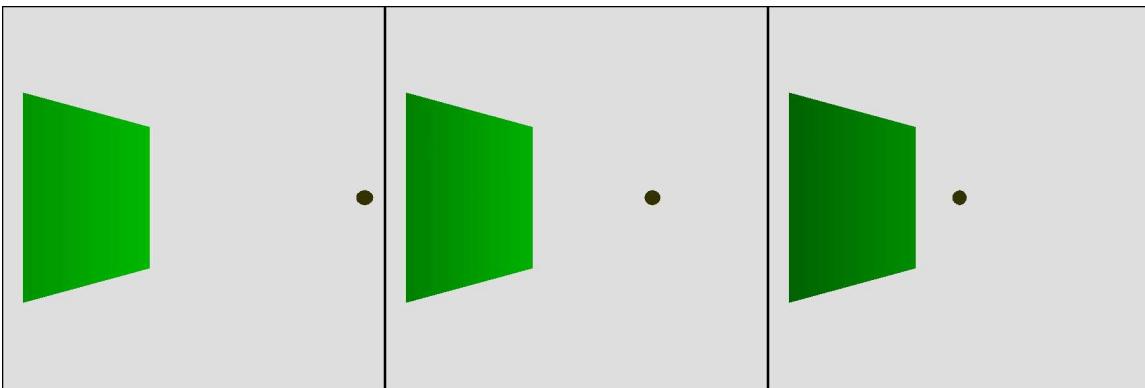


图 13-7：与我们的预期相反，点光源越靠近脸部，它看起来越暗。

我们期望三角形中心附近的点接收到大量的光，因为 \vec{L} 和 \vec{N} 大致平行。但是，我们不是在三角形的中心计算照明，而是在其顶点处计算照明。在那里，光线离表面越近，与法线的角度就越大，因此它们接收的照明很少。这意味着每个内部像素最终都会得到一个强度值，该值是在两个小值之间进行插值的结果，这也是一个低值，如图 13-8 所示。

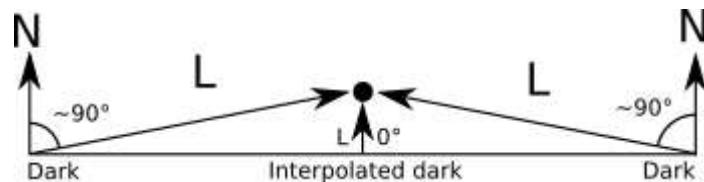


图 13-8：来自暗顶点的插值照明会产生暗中心，尽管法线与该点的光矢量平行。

那么，该怎么办呢？

蓬遮阳

我们可以克服 Gouraud 着色的局限性，但像往常一样，在质量和资源使用之间需要权衡。

平面阴影涉及每个三角形的单个照明计算。Gouraud 着色需要为每个三角形进行三次照明计算，外加跨三角形的单个属性（照明）的插值。质量的下一步要求我们计算三角形每个像素的照明。

从理论的角度来看，这听起来并不特别复杂；我们已经在计算一三个点的照明，毕竟我们正在计算光线追踪器的每像素照明。这里棘手的是弄清楚照明方程的输入来自哪里。回想一下，包含环境、漫反射和镜面反射分量的完整照明方程为：

$$I_P = I_A + \sum_{i=1}^n I_i \left(\frac{\langle \vec{N}, \vec{L}_i \rangle}{|\vec{N}| |\vec{L}_i|} + \left(\frac{\langle \vec{R}, \vec{V} \rangle}{|\vec{R}| |\vec{V}|} \right)^s \right)$$

首先，我们需要 \vec{L} 。对于定向光， \vec{L} 被给出。对于点光源， \vec{L} 定义为场景中点的矢量， P ，到光的位置， Q 。但是，我们没有 P 对于三角形的每个像素，但仅适用于顶点。

我们所拥有的是 P ；也就是说， x' 和 y' 我们即将在画布上绘画！我们知道

$$x' = \frac{xd}{z}$$

$$y' = \frac{yd}{z}$$

我们也碰巧有一个插值但几何上正确的值 $\frac{1}{z}$ 作为深度缓冲算法的一部分，因此

$$x' = xd\frac{1}{z}$$

$$y' = yd\frac{1}{z}$$

我们可以恢复 P 从这些值：

$$x = \frac{x'}{d\frac{1}{z}}$$

$$y = \frac{y'}{d\frac{1}{z}}$$

$$z = \frac{1}{\frac{1}{z}}$$

我们还需要 \vec{V} . 这是从相机（我们知道）到 P （我们刚刚计算），所以 \vec{V} 只是 $P - C$.

接下来，我们需要 \vec{N} . 我们只知道三角形顶点处的法线。当你只有一把锤子时，每个问题看起来都像钉子；我们的锤子是——你可能猜到了——属性值的线性插值。因此，让我们取以下值 N_x, N_y 和 N_z 在每个顶点，并将每个顶点视为我们可以线性插值的属性。然后，在每个像素上，我们将插值的分量重新组装成一个向量，对其进行归一化，并将其用作该像素的法线。

这种技术被称为 *Phong* 阴影，以 Bui Tuong Phong 的名字命名，他在 1973 年发明了它。图 13-9 显示了结果。

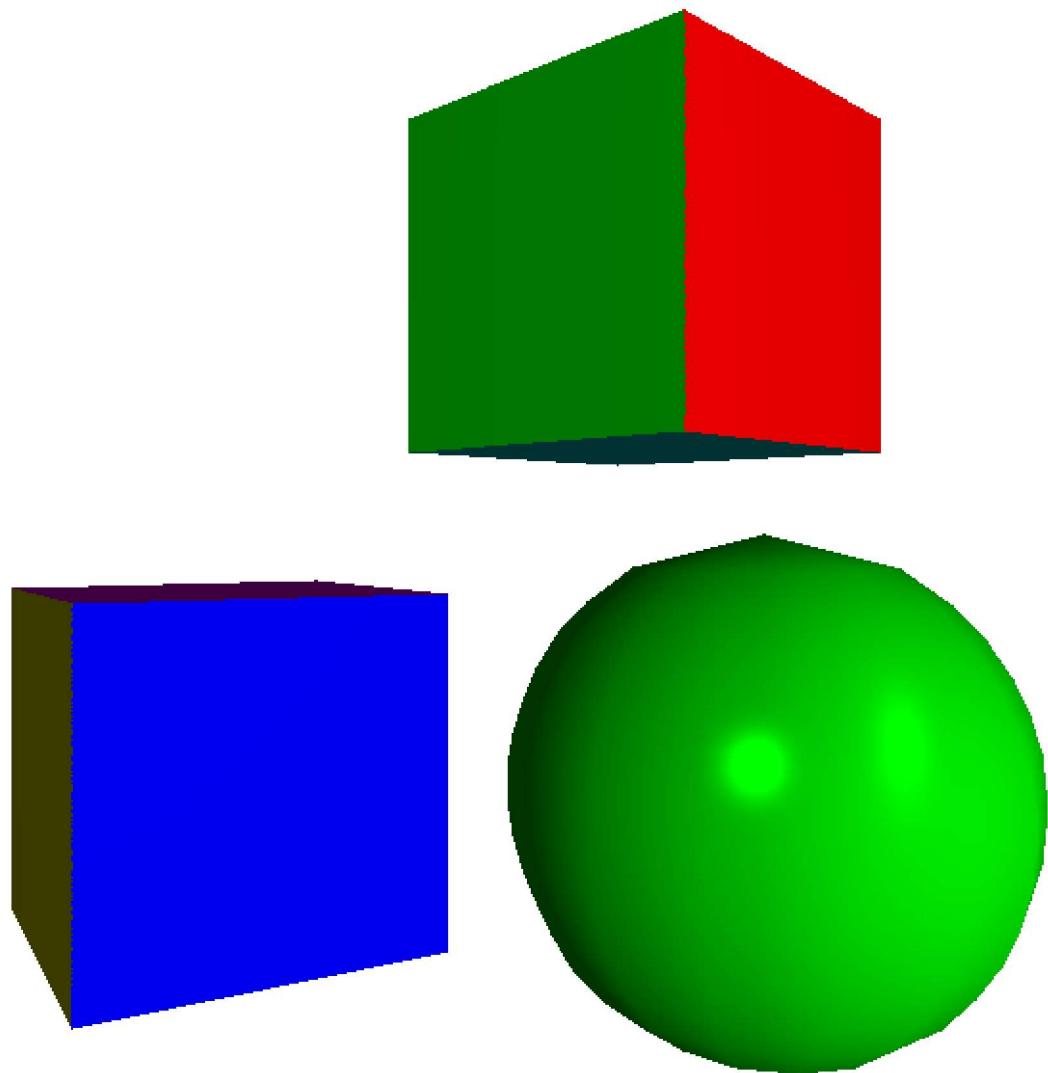


图 13-9: *Phong* 阴影。球体表面看起来很光滑，镜面反射高光清晰可见。

[源代码和现场演示>>](#)

球体现在看起来好多了。其表面显示适当的曲率，并且镜面反射高光看起来很清晰。然而，轮廓仍然背叛了这样一个事实，即我们正在渲染由三角形组成的近似值。这不是着色算法的缺点，它只确定三角形表面每个像素的颜色，但无法控制三角形本身的形式。这个球体近似使用 420 个三角形；我们可以通过使用更多的三角形来获得更平滑的轮廓，但代价是性能更差。

Phong阴影还解决了光线接近面部的问题，现在给出了预期的结果（图13-10）。

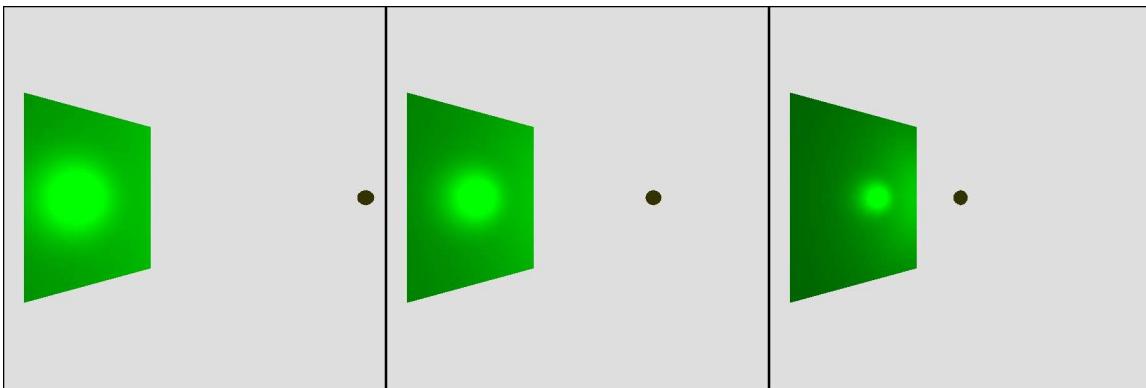


图 13-10：光线离表面越近，镜面反射高光外观越亮、越清晰。

在这一点上，我们已经匹配了第一部分中开发的光线追踪器的功能，除了阴影和反射。使用完全相同的场景定义，图 13-11 显示了我们正在开发的光栅器的输出。

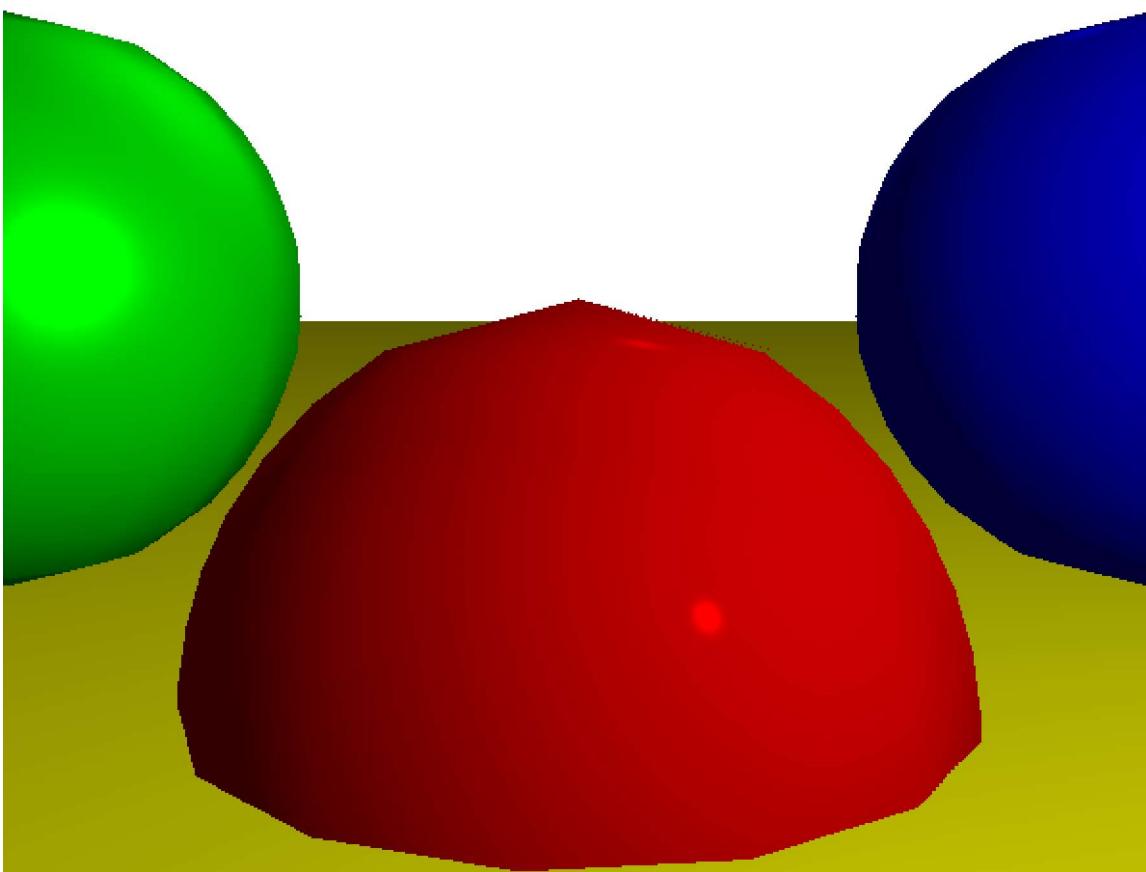


图 13-11：由光栅器渲染的参考场景

作为参考，图 13-12 显示了同一场景的光线跟踪版本。

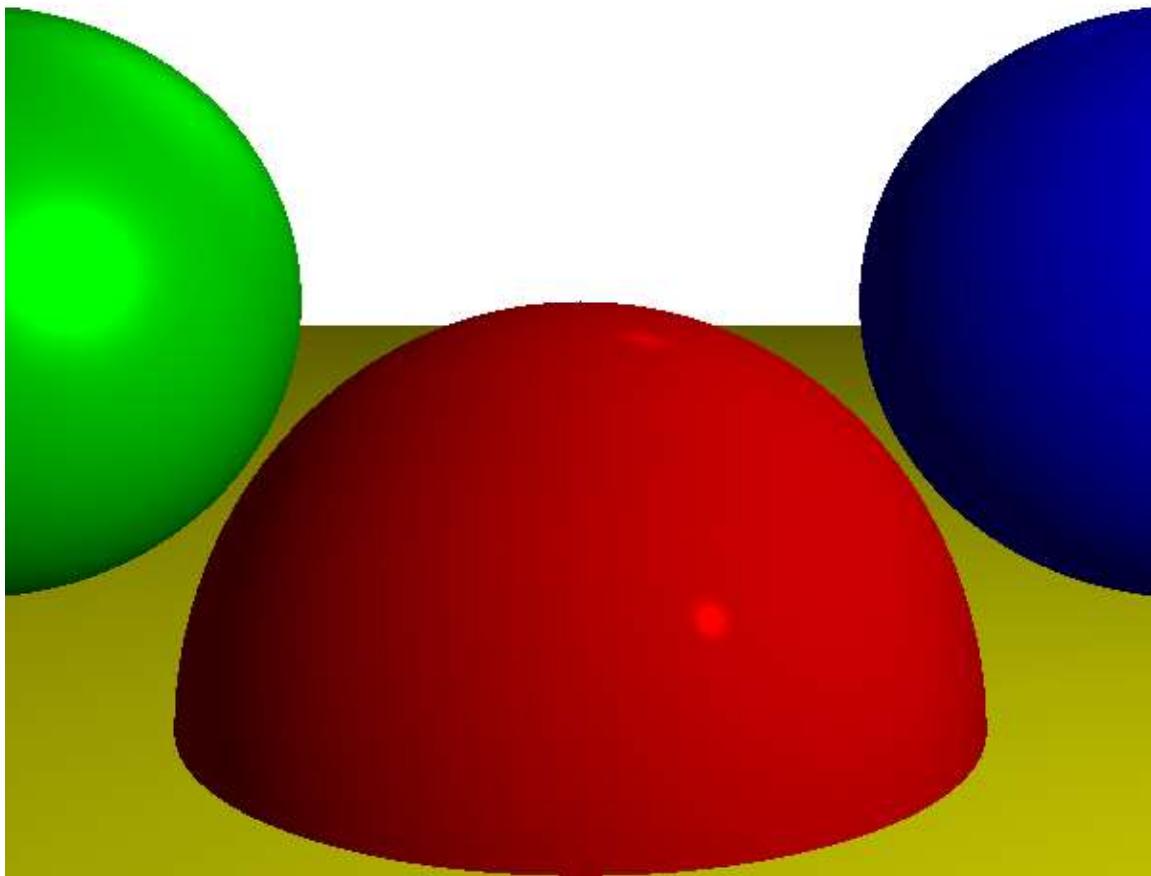


图 13-12：光线追踪器渲染的参考场景

这两个版本看起来几乎相同，尽管使用了截然不同的技术。这是意料之中的，因为场景定义是相同的。唯一可见的区别可以在球体的轮廓中找到：光线追踪器将它们渲染为数学上完美的对象，但我们使用由三角形组成的近似值作为光栅器。

另一个区别是两个渲染器的性能。这非常依赖于硬件和实现，但一般来说，光栅器可以每秒生成多达 60 次或更多的复杂场景的全屏图像，这使得它们适用于视频游戏等交互式应用程序，而光线追踪器可能需要几秒钟才能渲染一次相同的场景。这种差异将来可能会消失；近年来硬件的进步使光线追踪器的性能与光栅仪相比更具竞争力。

总结

在本章中，我们为光栅器添加了照明。我们使用的照明方程与[第 3 章（光源）](#)中的公式完全相同，因为我们使用相同的照明模型。但是，光线追踪器计算每个像素的照明方程，而我们的光栅器可以支持各种不同的技术，以实现性能和图像质量之间的特定权衡。

最快的着色算法（也会产生最不吸引人的结果）是平面着色：我们计算三角形中单个点的照明，并将其用于该三角形中的每个像素。这会产生非常多面的外观，特别是对

于近似曲面的对象（如球体）。

在质量阶梯上更进一步，我们有 Gouraud 着色：我们计算三角形三个顶点的照明，然后将该值插值到三角形的表面上。这使对象（包括弯曲的对象）具有更平滑的外观。但是，此技术无法捕捉到更微妙的照明效果，例如镜面反射高光。

最后，我们研究了Phong阴影。与我们的光线追踪器非常相似，它可以计算每个像素的照明方程，从而产生最佳结果和最差的性能。Phong阴影的诀窍是知道如何计算所有必要的值来评估照明方程；同样，答案是线性插值 - 在这种情况下，是法线向量。

在下一章中，我们将使用一种我们尚未研究过光线追踪器的技术：纹理映射，为三角形的表面添加更多细节。