

光

我们将通过引入光线开始在场景渲染中添加“真实感”。光是一个庞大而复杂的主题，因此我们将提供一个简化的模型，该模型足以满足我们的目的。该模型在很大程度上受到现实世界中光线工作方式的启发，但为了使渲染的场景看起来不错，它也需要一些自由。

我们将从一些简化的假设开始，这将使我们的生活更轻松，然后我们将介绍三种类型的光源：点光源、定向光和环境光。在本章结束时，我们将讨论这些光源如何影响表面的外观，包括漫反射和镜面反射。

简化假设

让我们做一些假设，让事情变得更简单。首先，我们声明所有的光都是白色的。这让我们可以使用单个实数 i 来表征任何光，称为光的强度。模拟彩色灯光并不复杂（我们只使用三个强度值，每个颜色通道一个，并按通道计算所有颜色和照明），但为了简单起见，我们将坚持使用白光。

其次，我们将忽略气氛。在现实生活中，灯光越远看起来越暗；这是因为漂浮在空气中的粒子在穿过它们时吸收了部分光。虽然这在光线追踪器中并不是特别复杂，但我们会保持简单并忽略这种效果；在我们的场景中，距离不会降低灯光的亮度。

光源

光必须来自某个地方。在本节中，我们将定义三种不同类型的光源。

点光源

点光源从 3D 空间中的固定点（称为其位置）发光。它们向各个方向均匀发光；这就是为什么它们也被称为全向灯。因此，点光源可以通过其位置和强度来完全描述。

灯泡是点光源的良好现实近似值。虽然现实生活中的灯泡不会从单个点发光，也不是完全全向的，但这是一个相当准确的近似值。

让我们将向量 \vec{L} 定义为从场景中的点 P 到光源 Q 的方向。我们可以计算这个矢量，称为光矢量，作为 $Q - P$ 。请注意，由于 Q 是固定的，但 P 可以是场景中的任何点，因此 \vec{L} 对于场景中的每个点都是不同的，如图 3-1 所示。

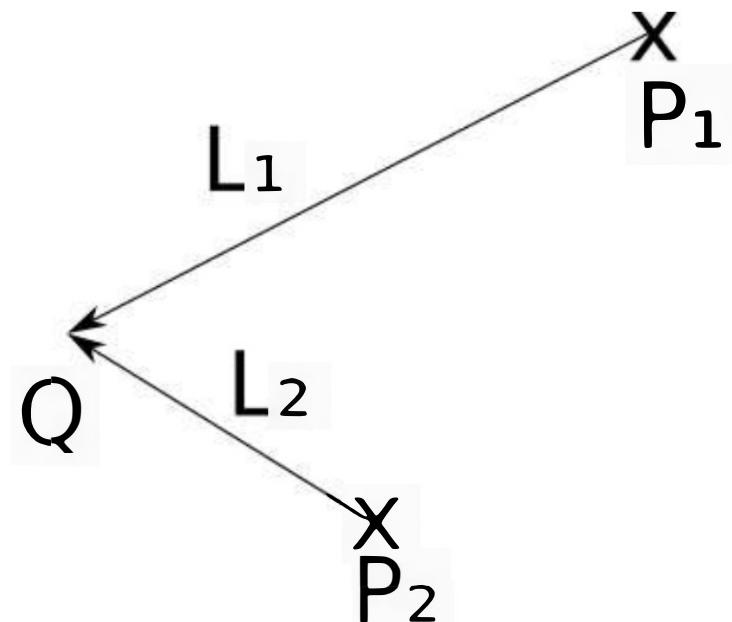


图 3-1: Q 处的点光源。向量 \vec{L} 对于每个点 P 都是不同的。

定向光

如果点光源是灯泡的良好近似值，那么它是否也可以作为太阳的近似值？

这是一个棘手的问题，答案取决于我们试图渲染的内容。在太阳系尺度上，太阳可以近似为点光。毕竟，它从一个点发光，它向各个方向发射，所以它似乎符合条件。

但是，如果我们的场景代表地球上发生的事情，那就不是一个很好的近似值。太阳是如此遥远，以至于到达我们的每一束光几乎都有完全相同的方向。我们可以用一个离场景中物体非常、非常、非常远的点光源来近似这一点。然而，光和物体之间的距离会比物体之间的距离大几个数量级，所以我们开始遇到数值精度错误。

为了更好地处理这些情况，我们定义了定向光。与点光源一样，定向光源具有强度但与它们不同的是，它们没有位置；相反，它们有一个固定的方向。您可以将它们视为位于指定方向的无限远点光源。

在点光源的情况下，我们需要为场景中的每个点 P 计算不同的光矢量 \vec{L} ，在这种情况下，给出了 \vec{L} 。在太阳到地球场景示例中， \vec{L} 将是（太阳的中心） – （地球中心）。图 3-2 显示了它的外观。

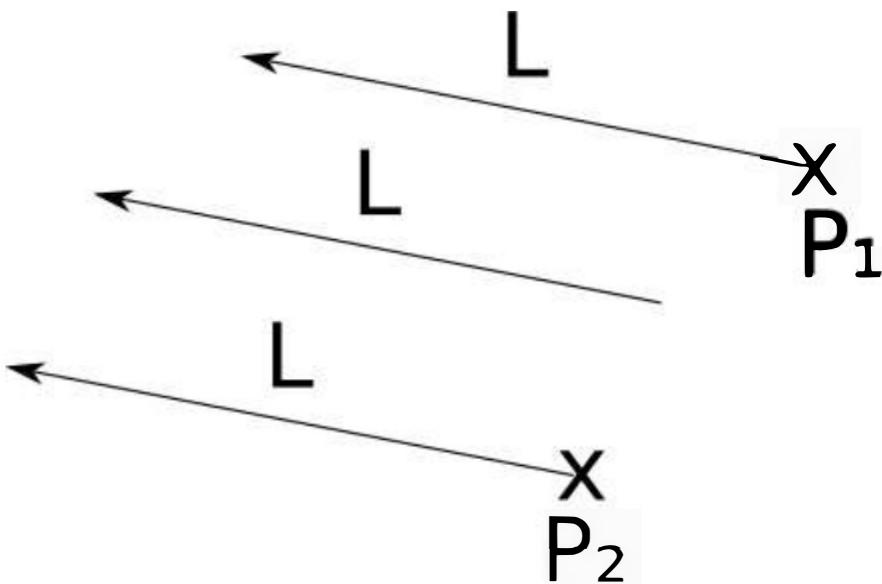


图 3-2: 定向光。 \vec{L} 向量对于每个点 P 都是相同的。

正如我们在这里看到的，定向光的光矢量对于场景中的每个点都是相同的。将其与图 3-1 进行比较，其中点光源的光矢量对于场景中的每个点都不同。

环境光

可以将每个现实生活中的光建模为点光或定向光吗？几乎。这两种类型的光线足以照亮场景吗？不幸的是不是。

考虑一下月球会发生什么。附近唯一重要的光源是太阳。因此，月球相对于太阳的“前半部分”得到了所有的光，而它的“后半部分”则处于完全黑暗的状态。我们从地球的不同角度看到这一点，创造了我们所说的月球“相位”。

然而，地球上的情况有点不同。即使是不直接从光源接收光线的点也不是完全处于黑暗中（看看椅子下面的地板）。如果光线对光源的“视野”被其他东西挡住，它们如何到达这些点？

如第1章（[介绍性概念](#)）的“颜色模型”中所述，当光线照射到物体上时，其中一部分被吸收，但其余部分被散射回场景中。这意味着光不仅可以来自光源，还可以来自光源获取光并将其部分散射回场景中的物体。但为什么要止步于此呢？散射光会反过来照射到其他物体上，一部分会被吸收，一部分会被散射回场景中。依此类推，直到原始光的所有能量都被场景中的表面吸收。

这意味着我们应该将每个物体都视为光源。你可以想象，这会给我们的模型增加很多复杂性，所以我们不会在本书中探讨这种机制。如果您好奇，请搜索全局照明并惊叹于美丽的图片。

但我们仍然不希望每个物体都被直接照亮或完全黑暗（除非我们真的渲染了太阳系的模型）。为了克服这一限制，我们将定义第三种类型的光源，称为环境光，其特征仅在于其强度。我们将声明环境光为场景中的每个点提供一些光，无论它位于何处。这是对场景中光源和表面之间非常复杂的交互的严重简化，但它的效果很好。

通常，场景将具有单个环境光（因为环境光只有一个强度值，因此任意数量的环境光都可以简单地组合成单个环境光）以及任意数量的点光源和定向光源。

单点照明

现在我们知道了如何定义场景中的光源，我们需要弄清楚光源如何与场景中对象的表面交互。

为了计算单个点的照明，我们将计算每个光源贡献的光量，并将它们相加以得到一个数字，表示该点接收的光总量。然后，我们可以将该点的表面颜色乘以此量，得到表示它接收多少光的颜色阴影。

那么，当一束光（无论是来自定向光还是点光源）照射到场景中某个物体上的某个点时会发生什么？

我们可以根据物体反射光线的方式直观地将物体分为两大类：“哑光”和“闪亮”物体。由于我们周围的大多数物体都可以归类为遮罩，因此我们将首先关注这一组。

漫反射

当光线照射到遮罩物体上时，光线会均匀地向各个方向散射回场景中，这一过程称为漫反射，这就是使遮罩对象看起来哑光的原因。

要验证这一点，请查看您周围的一些遮罩对象，例如墙壁。如果相对于墙壁移动，其颜色不会改变。也就是说，无论您从哪里看，您从物体反射的光都是相同的。

另一方面，反射的光量确实取决于光线与表面之间的角度。直观地说，这是因为光线携带的能量必须根据角度分布在更小或更大的区域内，因此每单位面积反射到场景的能量分别更高或更低，如图3-3所示。

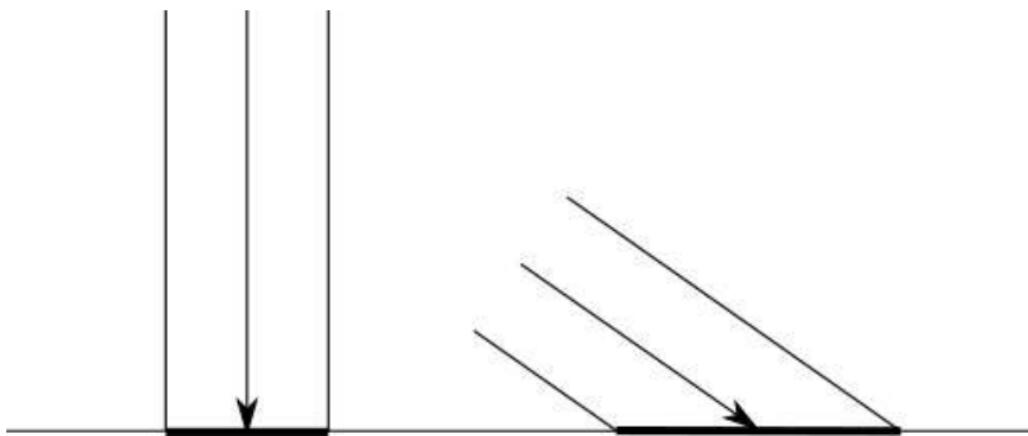


图 3-3：光线的能量散布在不同大小的区域，具体取决于其与表面的角度。

在图 3-3 中，我们可以看到两束相同强度（以相同宽度表示）的光线正面以一定角度照射到表面。光线携带的能量均匀地分布在它们击中的区域。右侧射线的能量分布比左侧射线的能量分布在更大的区域内，因此其区域内的每个点接收的能量都比左侧的情况少。

为了从数学上探索这一点，让我们通过其法线向量来表征表面的方向。点P处的表面的法线向量，或简称“法线”，是垂直于点P处的表面的向量。它也是一个单位向量，这意味着它的长度是1。我们将这个向量称为 \vec{N} 。

漫反射建模

有方向的光线为 \vec{L} 和强度为 I 的光线照射到法线 \vec{N} 的表面。

I 的哪一部分作为 I 、 \vec{N} 和 \vec{L} 的函数反射回场景？

作为几何类比，让我们将光的强度表示为光线的“宽度”。它的能量分布在大小为 A 的表面上。当 \vec{N} 和 \vec{L} 具有相同的方向时 - 当射线垂直于表面时 - 则 $I = A$ ，这意味着每单位面积反射的能量与每单位面积的入射能量相同： $\frac{I}{A} = 1$ 。另一方面，当 \vec{L} 和 \vec{N} 之间的角度接近 90° 时， A 接近 ∞ ，所以每单位面积的能量接近 0 ； $A \rightarrow \infty$ 时， $\frac{I}{A} = 0$ 。但是这两者之间会发生什么？

情况如图 3-4 所示。我们知道 \vec{N} 、 \vec{L} 和 P ；我添加了角度 α 和 β ，以及点 Q 、 R 和 S ，以便更轻松地编写图表。

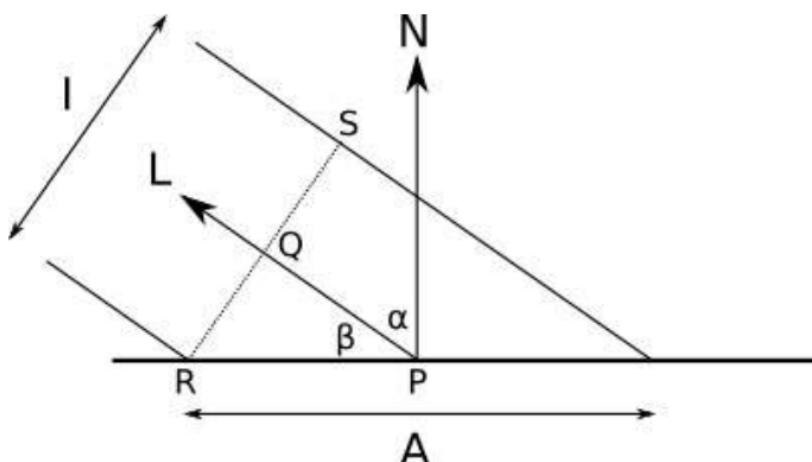


图 3-4：漫反射计算中涉及的矢量和角度

由于光线在技术上没有宽度，我们可以假设一切都发生在一个平坦的、无限小的表面。即使它是一个球体的表面，我们正在考虑的面积也非常小，与球体的大小相比，它几乎是平坦的，就像地球在小尺度下看起来是平坦的一样。

宽度为 I 的光线以 P 的角度照射到表面。 P 处的法线为 \vec{N} ，光线携带的能量扩散在 A 上。我们需要计算 $\frac{I}{A}$ 。

考虑 RS ，即射线的“宽度”。根据定义，它垂直于 \vec{L} ，这也是 PQ 的方向。因此， PQ 和 QR 形成一个直角，使 PQR 成为一个直角三角形。

PQR 的一个角度是 90° ，另一个角度是 β 。因此，剩余的角度为 $90^\circ - \beta$ 。但请注意， \vec{N} 和 PR 也形成一个直角，这意味着 $\alpha + \beta$ 也必须是 90° 。因此， $\widehat{QRP} = \alpha$ 。

让我们专注于三角形 PQR （图 3-5）。它的角度是 α 、 β 和 90° 。侧 QR 测量 $\frac{I}{2}$ ，侧 PR 测量 $\frac{A}{2}$ 。

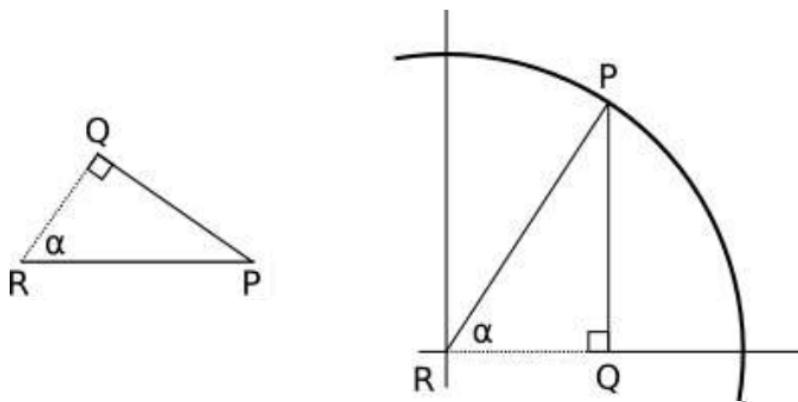


图 3-5: 三角函数上下文中的 PQR 三角形

现在，三角学来救援！根据定义， $\cos(\alpha) = \frac{QR}{PR}$ ；将 QR 替换为 $\frac{I}{2}$ ，将 PR 替换为 $\frac{A}{2}$ ，我们将得到

$$\cos(\alpha) = \frac{\frac{I}{2}}{\frac{A}{2}}$$

这成为

$$\cos(\alpha) = \frac{I}{A}$$

我们快到了。 α 是 \vec{N} 和 \vec{L} 的角度。我们可以使用点积的性质（随意查阅线性代数附录）将 $\cos(\alpha)$ 表示为

$$\cos(\alpha) = \frac{\langle \vec{N}, \vec{L} \rangle}{|\vec{N}| |\vec{L}|}$$

最后

$$\frac{I}{A} = \frac{\langle \vec{N}, \vec{L} \rangle}{|\vec{N}| |\vec{L}|}$$

我们得出了一个简单的方程，它给出了反射的光的比例，该比例是表面法线与光的方向之间角度的函数。

请注意， $\cos(\alpha)$ 的值对于超过 90° 的角度变为负数。如果我们盲目地使用这个值，我们最终会得到一个使表面更暗的光源！这没有任何物理意义；超过 90° 的角度只是意味着光线实际上照亮了表面的背面，因此它不会为我们正在照亮的点提供任何光。因此，如果 $\cos(\alpha)$ 变为负数，我们需要将其视为 0。

漫反射方程

现在，我们可以公式化一个方程来计算法线为 \vec{N} 的点 P 在具有强度为 I_A 的环境光和 n 个点的场景中，或者具有强度 I_n 和光矢量 \vec{L}_n 的定向光已知（对于定向光）或针对 计算 P （对于点光源）

$$I_P = I_A + \sum_{i=1}^n I_i \frac{\langle \vec{N}, \vec{L}_i \rangle}{|\vec{N}| |\vec{L}_i|}$$

值得重复的是，不应将 $\langle \vec{N}, \vec{L}_i \rangle < 0$ 添加到点的照明中的项。

球体法线

只缺少一个小细节：法线从何而来？这个普遍问题的答案远比看起来要复杂得多，正如我们将在本书的第二部分看到的那样。幸运的是，在这一点上，我们只处理球体，并且有一个非常简单的答案：球体任何点的法向量位于穿过球体中心的线上。如图 3-6 所示，如果球心为 C ，则点 P 处的法线方向为 $P - C$ 。

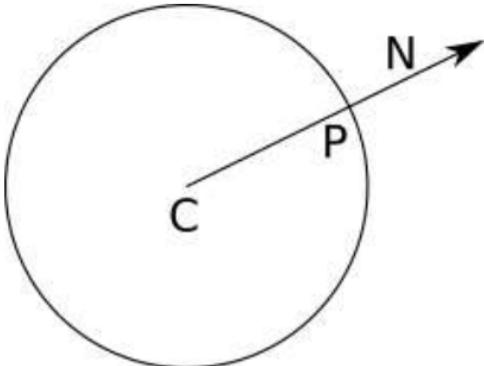


图 3-6：球体在 P 处的法线与 CP 的方向相同。

为什么是“正常的方向”而不是“正常”？法向量需要垂直于表面，但它也需要具有长度1。为了规范化这个向量并将其转换为真正的法线，我们需要将其除以它自己的长度，从而保证结果的长度为1：

$$\vec{N} = \frac{\vec{P} - \vec{C}}{|\vec{P} - \vec{C}|}$$

使用漫反射进行渲染

让我们将所有这些转换为伪代码。首先，让我们为场景添加几盏灯：

```
light {
    type = ambient
    intensity = 0.2
}
light {
    type = point
    intensity = 0.6
    position = (2, 1, 0)
}
light {
    type = directional
    intensity = 0.2
    direction = (1, 4, 4)
}
```

请注意，强度加起来很容易达到1.0；由于照明方程的工作方式，这可确保任何点的光强度都不能大于此值。这意味着我们不会有任何“过度曝光”的斑点。

照明方程相当容易转换为伪代码（示例 3-1）。

```
ComputeLighting(P, N) {
    i = 0.0
    for light in scene.Lights {
        if light.type == ambient {
```

```

    ❶i += light.intensity
} else {
    if light.type == point {
        ❷L = light.position - P
    } else {
        ❸L = light.direction
    }

    n_dot_l = dot(N, L)
    ❹if n_dot_l > 0 {
        ❺i += light.intensity * n_dot_l/(length(N) * length(L))
    }
}
return i
}

```

示例 3-1：使用漫反射计算光照的函数

在示例 3-1 中，我们以略微不同的方式处理这三种类型的光。环境光是最简单的，可以直接处理❶。点光源和定向光共享大部分代码，特别是强度计算 ❺，但方向矢量的计算方式不同（❷ 和 ❸），具体取决于其类型。❹ 中的条件确保我们不会添加负值，负值表示照亮表面背面的灯光，如前所述。

剩下唯一要做的就是在 中使用。我们替换返回球体颜色的行：ComputeLightingTraceRay

```
return closest_sphere.color
```

使用此代码片段：

```

P = O + closest_t * D // Compute intersection
N = P - closest_sphere.center // Compute sphere normal at intersection
N = N / length(N)
return closest_sphere.color * ComputeLighting(P, N)

```

只是为了好玩，让我们添加一个大的黄色球体：

```

sphere {
    color = (255, 255, 0) # Yellow
    center = (0, -5001, 0)
    radius = 5000
}

```

我们运行渲染器，瞧，球体现在开始看起来像球体（图 3-7）！

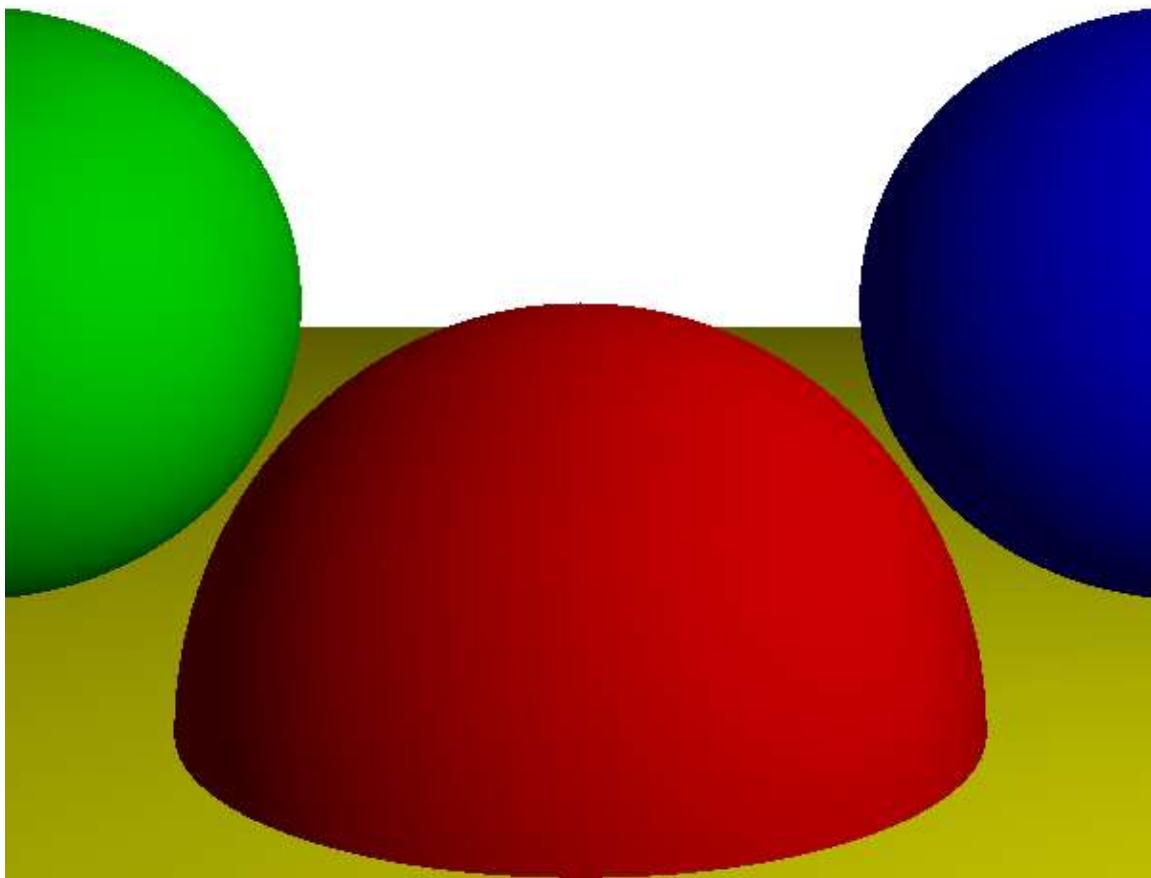


图 3-7：漫反射为场景增添了深度和体积感。

[源代码和现场演示>>](#)

但是等等，黄色的大球体是如何变成平坦的黄色地板的？它没有与其他三个球体相比，它是如此之大，而且相机离它如此之近，以至于它看起来很平坦——就像我们站在上面时，我们星球的表面看起来很平坦。

镜面反射

让我们把注意力转向闪亮的物体。与遮罩对象不同，闪亮对象的外观略有不同，具体取决于您从何处看。

想象一下，一个台球或一辆刚从洗车场出来的汽车。这些类型的物体表现出非常特殊的光模式，通常是亮点，当您在它们周围移动时，它们似乎会移动。与遮罩对象不同，您感知这些对象表面的方式实际上取决于您的观点。

请注意，如果你绕着它走，红色台球会保持红色，但赋予它闪亮外观的明亮白点会随着你的移动而移动。这表明我们想要建模的新效应并没有取代漫反射，而是补充了漫反射。

为了理解为什么会发生这种情况，让我们仔细看看表面如何反射光线。正如我们在上一节中看到的，当光线照射到遮罩物体的表面时，它会在各个方向上均匀地散射回场景。发生这种情况是因为物体的表面是不规则的，所以在微观层面上，它的行为就像一组指向随机方向的微小表面（图 3-8）：

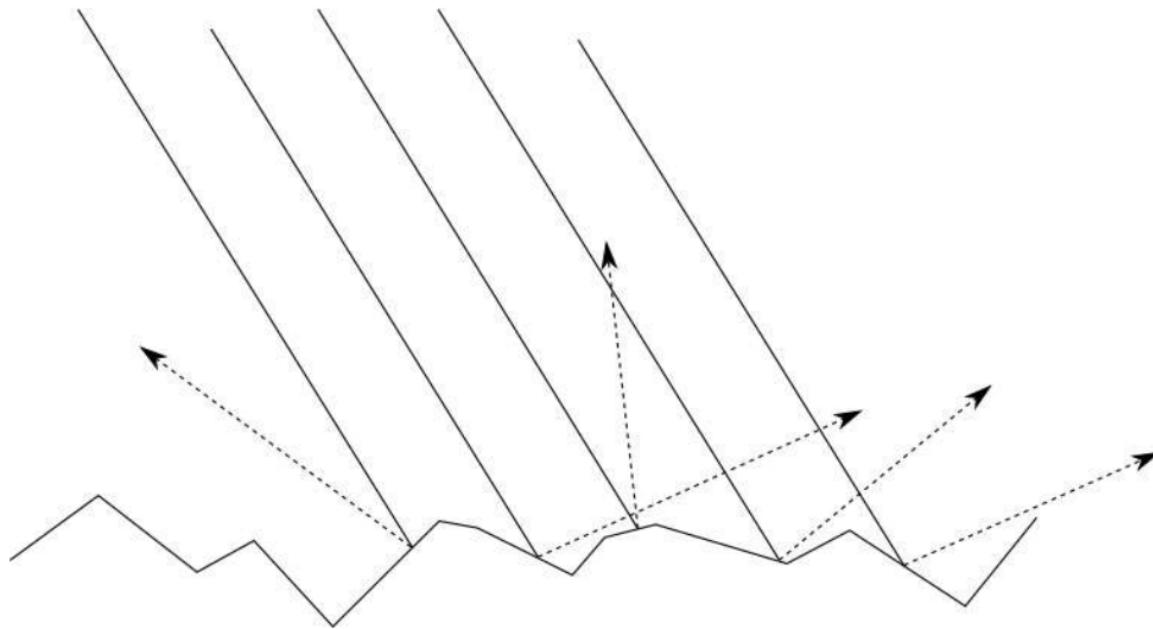


图 3-8：通过显微镜，哑光物体的粗糙表面可能是什么样子。入射光线以随机方向反射。

但是，如果表面不是那么不规则呢？让我们去另一个极端：一面完美抛光的镜子。当光线照射到镜子上时，它会向一个方向反射。如果我们调用反射光的方向 \vec{R} ，并保持 \vec{L} 指向光源的约定，图 3-9 说明了这种情况。

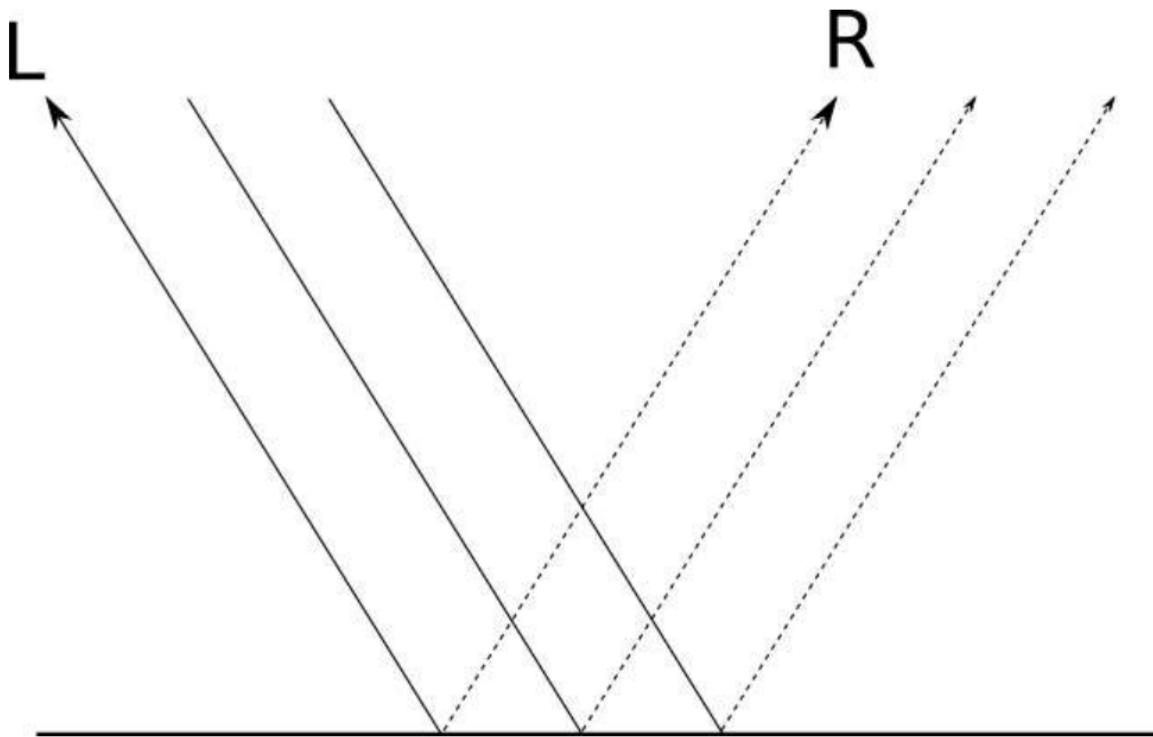


图 3-9：镜子反射的光线

根据表面的“抛光”程度，它的行为或多或少像一面镜子；这就是为什么它被称为镜面反射，来自窥镜，拉丁语中的镜子。

对于完美抛光的镜子，入射光线 \vec{L} 在单个方向 \vec{R} 上反射。这就是为什么你能非常清楚地看到反射物体的原因：对于每一个入射的光线 \vec{L} ，

都有一个反射光线 \vec{R} 。但并非每件物品都经过完美抛光；虽然大部分光在 \vec{R} 的方向上反射，但其中一些光在接近 \vec{R} 的方向上反射。离 \vec{R} 越近，在该方向上反射的光就越多，如图 3-10 所示。物体的

“光泽度”决定了当您远离 \vec{R} 时反射光减少的速度。

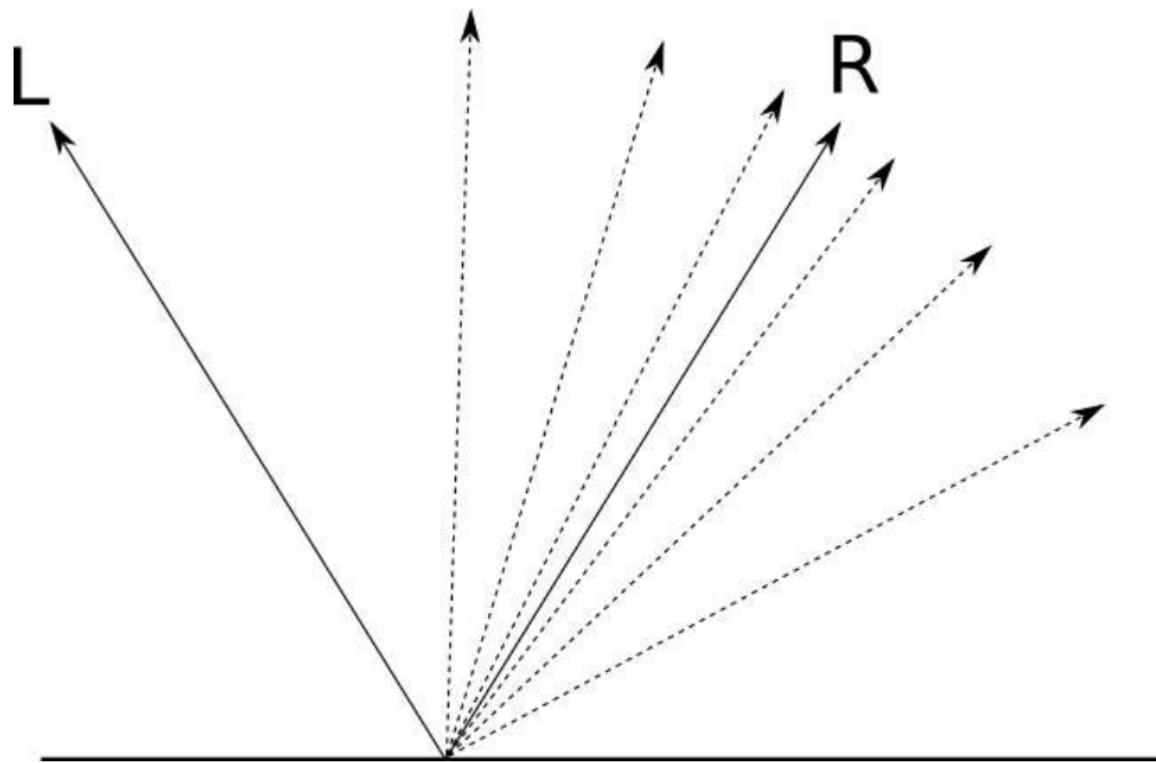


图 3-10：对于未完全抛光的表面，方向越接近 \vec{R} ，在该方向上反射的光线就越多。

我们想弄清楚有多少来自 \vec{L} 的光被反射回我们的观点方向。如果 \vec{V} 是从 P 指向相机的“视图向量”，而 α 是 \vec{R} 和 \vec{V} 之间的角度，我们得到图 3-11。

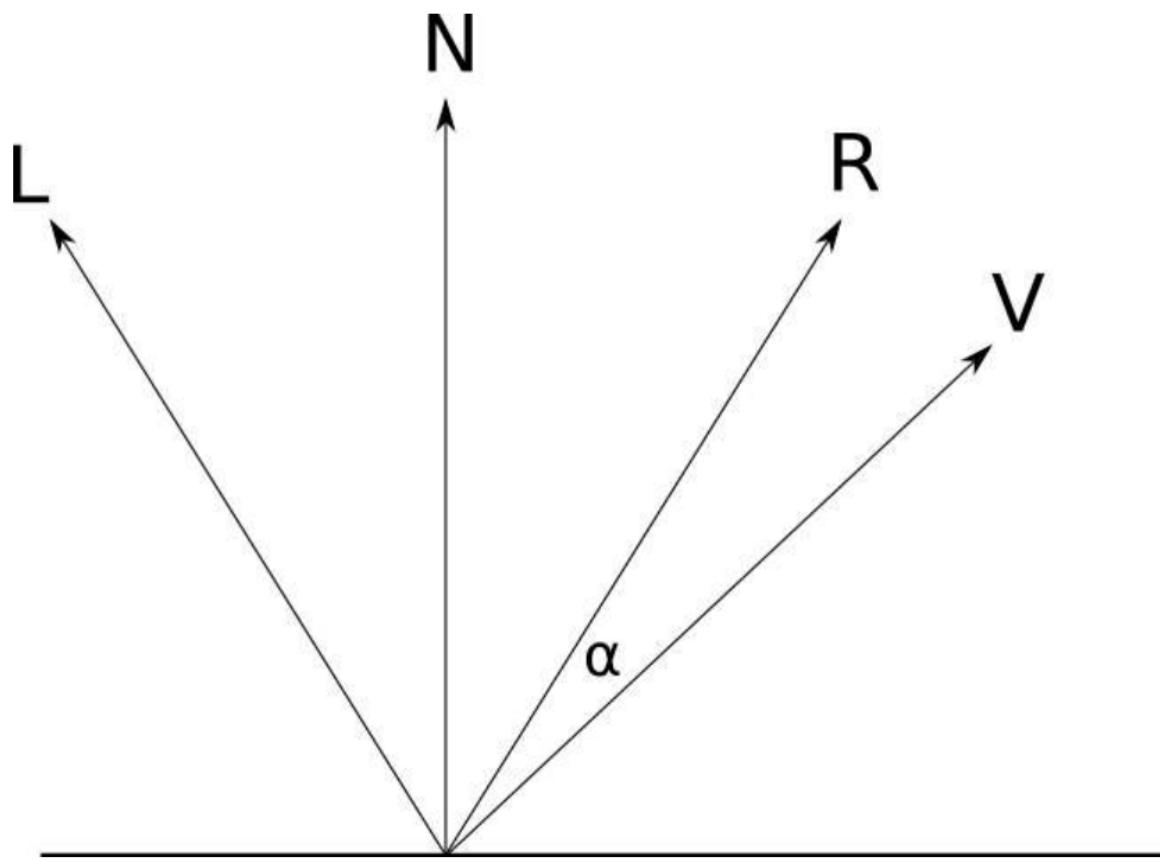


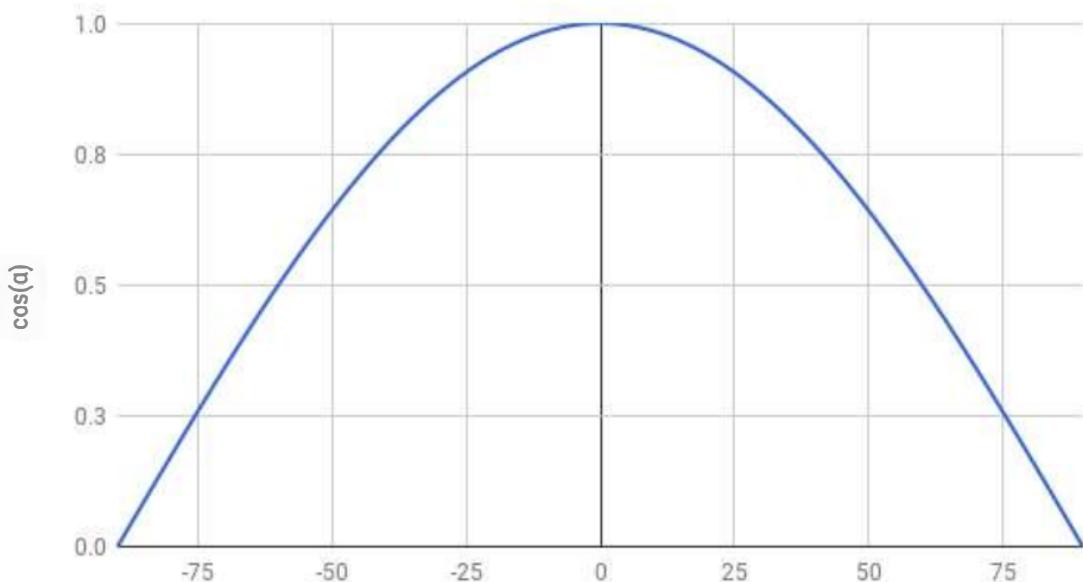
图 3-11：镜面反射计算中涉及的矢量和角度

对于 $\alpha = 0^\circ$ ，所有光都沿 \vec{V} 的方向反射。对于 $\alpha = 90^\circ$ ，不反射光。与漫反射一样，我们需要一个数学表达式来确定 α 的中间值会发生什么。

镜面反射建模

在本章开头，我提到有些模型不是基于物理模型的。这是其中之一。以下模型是任意的，但使用它是因为易于计算且看起来不错。

考虑 $\cos(\alpha)$ 。它具有 $\cos(0) = 1$ 和 $\cos(\pm 90) = 0$ 的良好属性，就像我们需要的那样；并且值在非常愉快的曲线中从 0 逐渐变小到 90° （图 3-12）。

图 3-12: $\cos(\alpha)$ 的图形。

这意味着 $\cos(\alpha)$ 符合我们对镜面反射函数的所有要求，那么为什么不使用它呢？

还有一个细节。如果我们直接使用这个公式，每个物体都会同样有光泽。我们如何调整方程来表示不同程度的光泽度？

请记住，光泽度是衡量反射函数随着 α 增加而减少的速度的指标。获得不同光泽度曲线的一种简单方法是计算 $\cos(\alpha)$ 与某个正指数 s 的幂。由于 $0 \leq \cos(\alpha) \leq 1$ ，我们保证 $0 \leq \cos(\alpha)^s \leq 1$ ；所以 $\cos(\alpha)^s$ 就像 $\cos(\alpha)$ 一样，只是“更窄”。图 3-13 显示了不同 s 值的 $\cos(\alpha)^s$ 的图形。

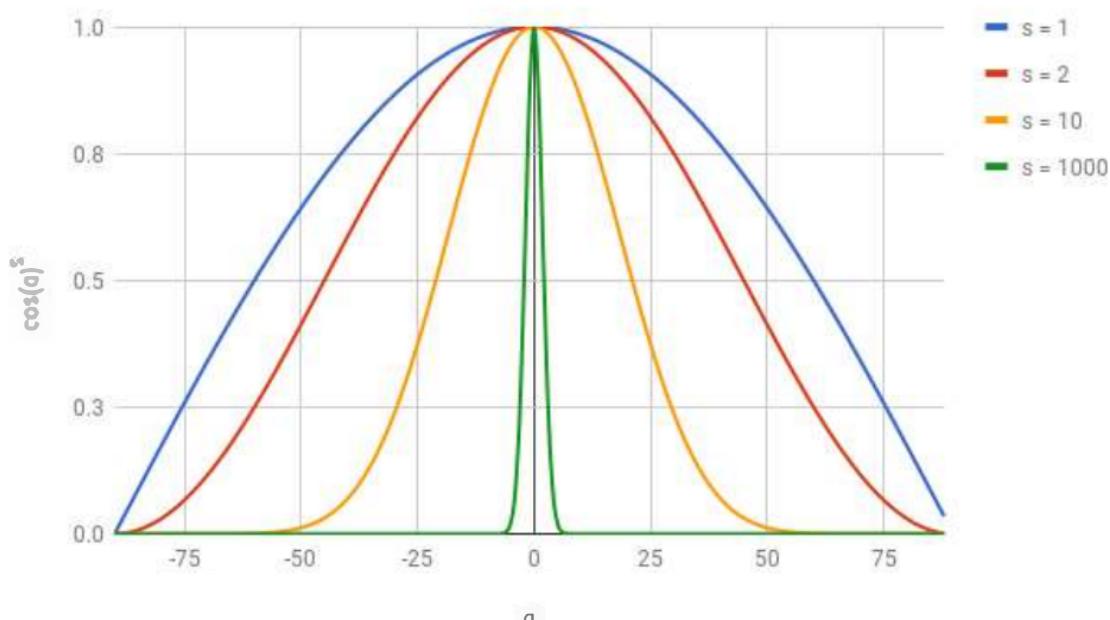


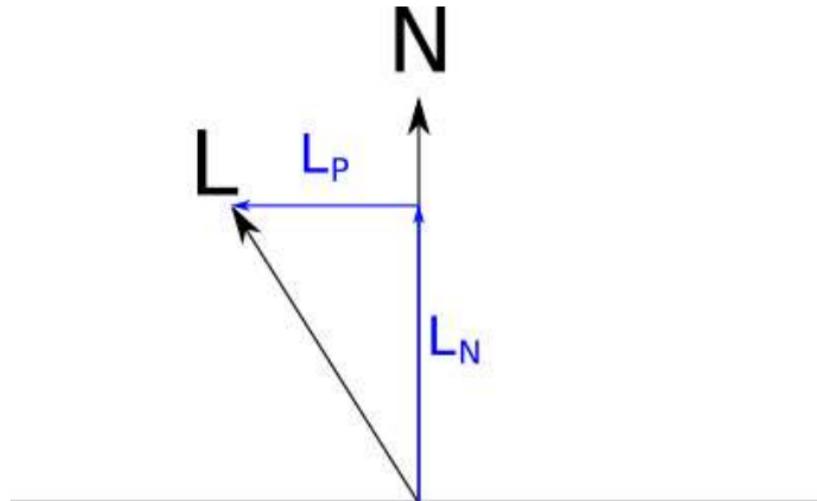
图 3-13: $\cos(\alpha)^s$ 的图形

s 的值越大，函数在 0 附近变得越“窄”，对象看起来越亮。 s 称为镜面反射指数，它是表面的一个属性。由于该模型不是基于物理现实，因此 s 的值只能通过反复试验来确定 - 本质上是调整值直到它们看起来“正确”。对于基于物理的模型，您可以查看双向反射函数（BDRF）。

让我们把所有这些放在一起。一束光线从方向 \vec{N} 照射到点 P 处具有镜面指数 s 的表面，其中其法线为 \vec{N} 。从方向 \vec{L} 来，有多少光被反射到观看方向 \vec{R} ？

根据我们的模型，这个值是 $\cos(\alpha)^s$ ，其中 α 是 \vec{V} 和 \vec{R} 之间的角度； \vec{R} 反过来 \vec{L} 相对于 \vec{N} 反映。因此，第一步是从 \vec{N} 和 \vec{L} 计算 \vec{R} 。

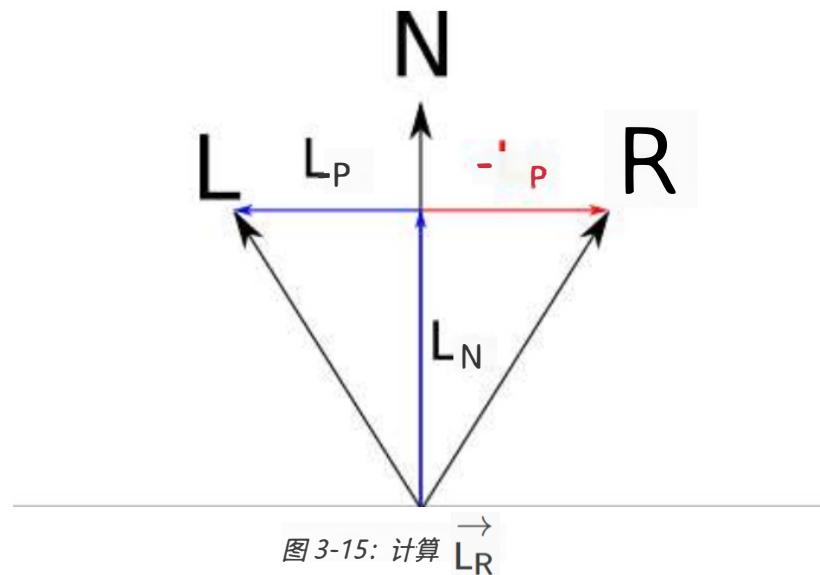
我们可以将 \vec{L} 分解为两个向量， \vec{L}_P 和 \vec{L}_N ，使得 $\vec{L} = \vec{L}_N + \vec{L}_P$ ，其中 \vec{L}_N 是平行 \vec{N} ，并且 \vec{L}_P 垂直于 \vec{N} （图 3-14）。

图 3-14: 将 \vec{L} 分解为其组件 \vec{L}_P 和 \vec{L}_N

\vec{L}_N 是 \vec{L} 在 \vec{N} 上的投影；根据点积的性质和 $|\vec{N}| = 1$ 的事实，此投影的长度为 $\langle \vec{N}, \vec{L} \rangle$ 。我们将 \vec{L}_N 定义为与 \vec{N} 并行，因此 $\vec{L}_N = \vec{N} \langle \vec{N}, \vec{L} \rangle$ 。

由于 $\vec{L} = \vec{L}_P + \vec{L}_N$ ，我们可以立即得到 $\vec{L}_P = \vec{L} - \vec{L}_N = \vec{L} - \vec{N} \langle \vec{N}, \vec{L} \rangle$

现在让我们看看 \vec{R} 。由于它相对于 \vec{N} 与 \vec{L} 对称，因此其平行于 \vec{N} 的分量与 \vec{L} 's 相同，其垂直分量与 \vec{L} 's 相反；也就是说 $\vec{R} = \vec{L}_N - \vec{L}_P$ 。您可以在图 3-15 中看到这一点。



用我们上面找到的表达式替换，我们得到

$$\vec{R} = \vec{N}\langle\vec{N}, \vec{L}\rangle - \vec{L} + \vec{N}\langle\vec{N}, \vec{L}\rangle$$

并简化一点

$$\vec{R} = 2\vec{N}\langle\vec{N}, \vec{L}\rangle - \vec{L}$$

镜面反射项

现在，我们准备为镜面反射编写一个方程：

$$\vec{R} = 2\vec{N}\langle\vec{N}, \vec{L}\rangle - \vec{L}$$

$$I_S = I_L \left(\frac{\langle \vec{R}, \vec{V} \rangle}{|\vec{R}| |\vec{V}|} \right)^s$$

与漫射照明一样， $\cos(\alpha)$ 可能是负数，出于与以前相同的原因，我们应该忽略它。此外，并非每个物体都必须有光泽；对于遮罩对象，根本不应计算镜面反射项。我们将在场景中通过将其镜面反射指数设置为 -1 并相应地处理它们来注意到这一点。

全照明方程

我们可以将镜面反射项添加到我们一直在开发的照明方程中，并获得描述某一点照明的单个表达式：

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

其中 I_P 是点 P 处的总照度， I_A 是环境光的强度， N 是 P 处表面的法线， V 是从 P 到相机的矢量， s 是表面的镜面指数， I_i 是光 i 的强度， L_i 是从 P 到光 i 的矢量， R_i 是光 i 在 P 处的反射矢量。

使用镜面反射进行渲染

让我们将镜面反射添加到目前为止一直在使用的场景中。首先，对场景本身进行一些更改：

```
sphere {
    center = (0, -1, 3)
    radius = 1
    color = (255, 0, 0) # Red
    specular = 500 # Shiny
}
sphere {
    center = (2, 0, 4)
    radius = 1
    color = (0, 0, 255) # Blue
    specular = 500 # Shiny
}
sphere {
    center = (-2, 0, 4)
    radius = 1
    color = (0, 255, 0) # Green
    specular = 10 # Somewhat shiny
}
sphere {
    center = (0, -5001, 0)
    radius = 5000
    color = (255, 255, 0) # Yellow
    specular = 1000 # Very shiny
}
```

这与以前的场景相同，只是在球体定义中添加了镜面反射指数。

在代码级别，我们需要在必要时更改以计算镜面反射项，并将其添加到整体光源中。请注意，该函数现在需要 \vec{V} 和 s ，如示例 3-2 所示。

ComputeLighting

```
ComputeLighting(P, N, V, s) {
    i = 0.0
    for light in scene.lights {
        if light.type == ambient {
            i += light.intensity
        } else {
            if light.type == point {
                l = light.position - P
            } else {
                L = light.direction
            }

            // Diffuse
            n_dot_l = dot(N, L)
            if n_dot_l > 0 {
                i += light.intensity * n_dot_l / (length(N) * length(L))
            }

            // Specular
            ❶ if s != -1 {
                R = 2 * N * dot(N, L) - L
                r_dot_v = dot(R, V)
                ❷ if r_dot_v > 0 {
                    i += light.intensity * pow(r_dot_v / (length(R) * length(V)), s)
                }
            }
        }
    }
}
```

```

    }
    return i
}

```

示例 3-2：支持漫反射和镜面反射 ComputeLighting

大多数代码保持不变，但我们添加一个片段来处理镜面反射。我们确保它仅适用于反光物体❶，并确保我们不会像漫反射那样添加负光强度❷。

最后，我们需要修改以将新参数传递给 s 。 s 很简单：它直接来自场景定义。但是 \vec{V} 从何而来？TraceRayComputeLighting

\vec{V} 是从对象指向相机的矢量。幸运的是，我们已经有一个从相机指向物体的矢量，即我们正在追踪的光线方向 \vec{D} ！所以 \vec{V} 就是简单的 $-\vec{D}$ 。

示例 3-3 给出了带有镜面反射的新内容。TraceRay

```

TraceRay(0, D, t_min, t_max) {
    closest_t = inf
    closest_sphere = NULL
    for sphere in scene.Spheres {
        t1, t2 = IntersectRaySphere(0, D, sphere)
        if t1 in [t_min, t_max] and t1 < closest_t {
            closest_t = t1
            closest_sphere = sphere
        }
        if t2 in [t_min, t_max] and t2 < closest_t {
            closest_t = t2
            closest_sphere = sphere
        }
    }
    if closest_sphere == NULL {
        return BACKGROUND_COLOR
    }

    P = 0 + closest_t * D // Compute intersection
    N = P - closest_sphere.center // Compute sphere normal at intersection
    N = N / length(N)
   ❶ return closest_sphere.color * Computelighting(P, N, -D, closest_sphere.specular)
}

```

示例 3-3：使用镜面反射 TraceRay

颜色计算❶比看起来稍微复杂一些。请记住，颜色必须逐通道相乘，结果必须固定在通道范围内（在本例中为[0–255]）。尽管在示例场景中，光照强度加起来为1.0，但现在我们添加了镜面反射的贡献，这些值可能会超出该范围。

您可以在图3-16中看到所有这些有效矢量的奖励。

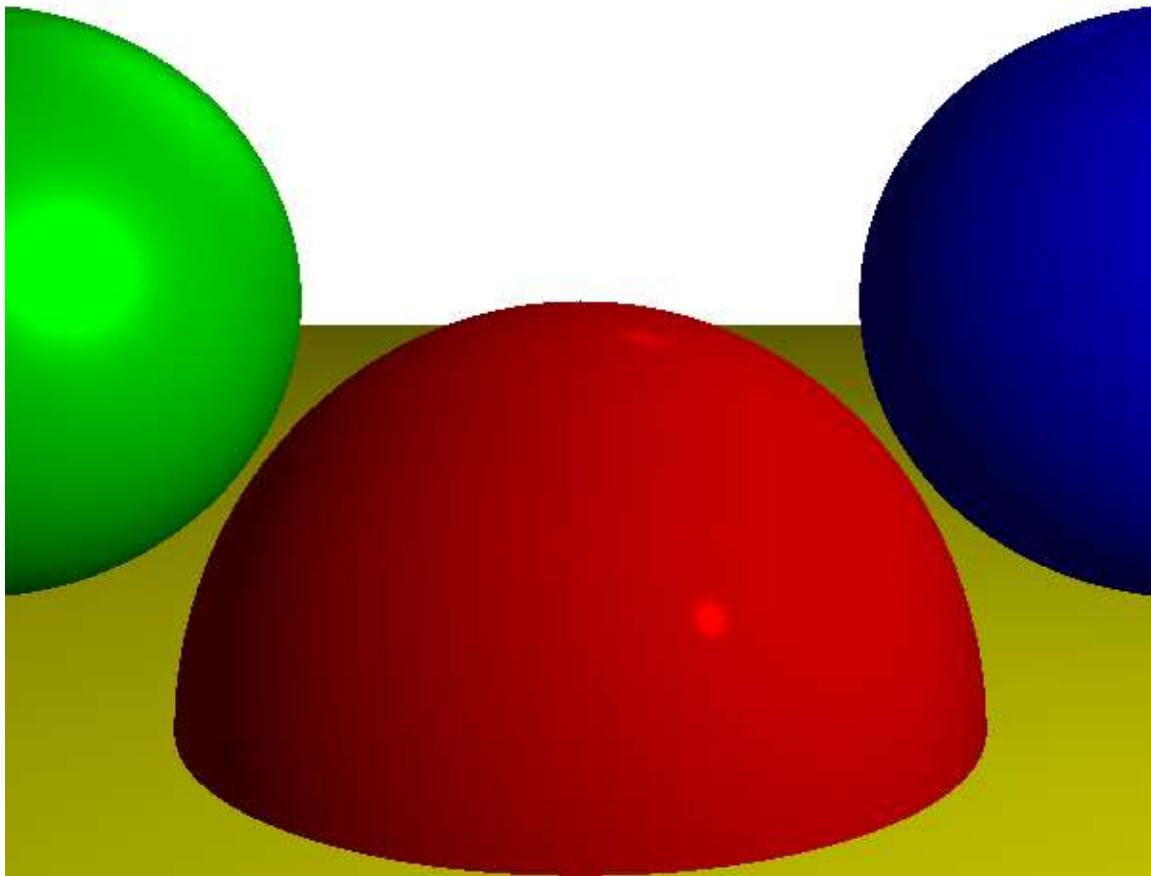


图 3-16：使用环境反射、漫反射和镜面反射渲染的场景。我们不仅获得了深度和体积感，而且每个表面的外观也略有不同。

[源代码和现场演示>>](#)

请注意，在图 3-16 中，镜面反射指数为 500 的红色球体比镜面指数为 10 的绿色球体具有更集中的亮点，与预期完全一致。蓝色球体的镜面反射指数也为 500，但没有可见的亮点。这只是图像裁剪方式和灯光在场景中放置方式的结果；事实上，红色球体的左半部分也没有表现出任何镜面反射。

总结

在本章中，我们采用了上一章中开发的非常简单的光线追踪器，并赋予它对灯光及其与场景中对象交互的方式进行建模的能力。

我们将灯光分为三种类型：点光源、定向光源和环境光源。我们探讨了它们中的每一个如何代表现实生活中可以找到的不同类型的光源，以及如何在我们的场景定义中描述它们。

然后，我们将注意力转向场景中物体的表面，将它们分为两种类型：哑光和闪亮。我们讨论了光线如何与它们相互作用，并开发了两个模型 - 漫反射和镜面反射 - 来计算

它们反射到相机的光量。

最终结果是场景渲染更加逼真：我们现在不再只看到物体的轮廓，而是获得真正的深度和体积感，以及对构成物体的材料的感觉。

然而，我们错过了灯光的一个基本方面：阴影。这是下一章的重点。