

裁剪

在最后几章中，我们开发了方程式和算法，将场景的 3D 定义转换为我们在画布上绘制的 2D 形状;我们开发了一个场景结构，让我们可以定义 3D 模型并将这些模型的实例放置在场景中;我们开发了一种算法，可以从任何角度渲染场景。

但是，这样做暴露了我们一直在使用的限制之一：透视投影方程仅对相机前方的点按预期工作。由于我们现在可以在场景中移动和旋转摄像机，这会带来一个问题。

在本章中，我们将开发消除此限制所需的技术：我们将探索如何识别相机后面的点、三角形和整个对象，并开发处理它们的技术。

剪辑过程概述

回到[第9章 \(透视投影\)](#)，我们得出了以下等式：

$$P'_x = \frac{P_x \cdot d}{P_z}$$

$$P'_y = \frac{P_y \cdot d}{P_z}$$

除以 P_z 是有问题的;它可能导致除以零。此外，相机后面的点具有负值 Z ，我们目前无法正确处理。即使是相机前方但非常靠近它的点也会以严重扭曲的物体的形式造成麻烦。

为了避免这些问题的情况，我们将选择不在投影平面 $Z=d$ 后面渲染任何内容。此剪裁平面允许我们将任何点分类为剪裁体积的内部或外部，即从相机实际可见的空间子集。在这种情况下，剪切体积是“ $Z=d$ 前面的任何内容”。我们将只渲染剪裁体积内的场景部分。

剪辑卷

使用单个剪切平面来确保不会渲染摄像机后面的对象将产生正确的结果，但这并不完全有效。有些物体可能在相机前，但仍然不可见;例如，投影平面附近但向右很远的对象的投影将投影到视口之外，因此不可见，如图 11-1 所示。

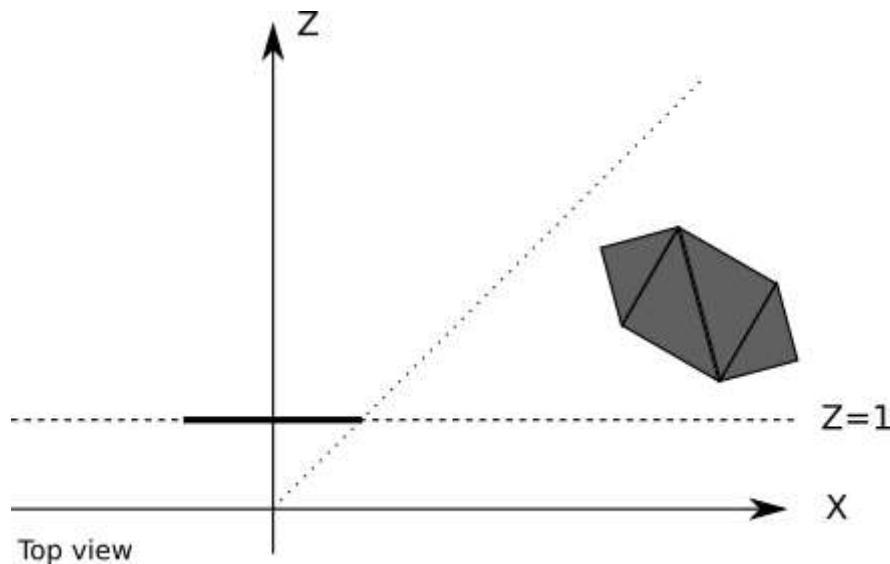


图 11-1：投影平面前面的对象，但将投影到视口之外

我们用来投影此类对象的任何计算资源，以及为渲染它而进行的所有每三角形和每顶点计算，都将被浪费。完全忽略这些对象会更有效。

为此，我们可以定义其他平面，将场景裁剪为视口上应该可见的平面；这些平面由相机和视口的四个边中的每一个定义（图 11-2）。

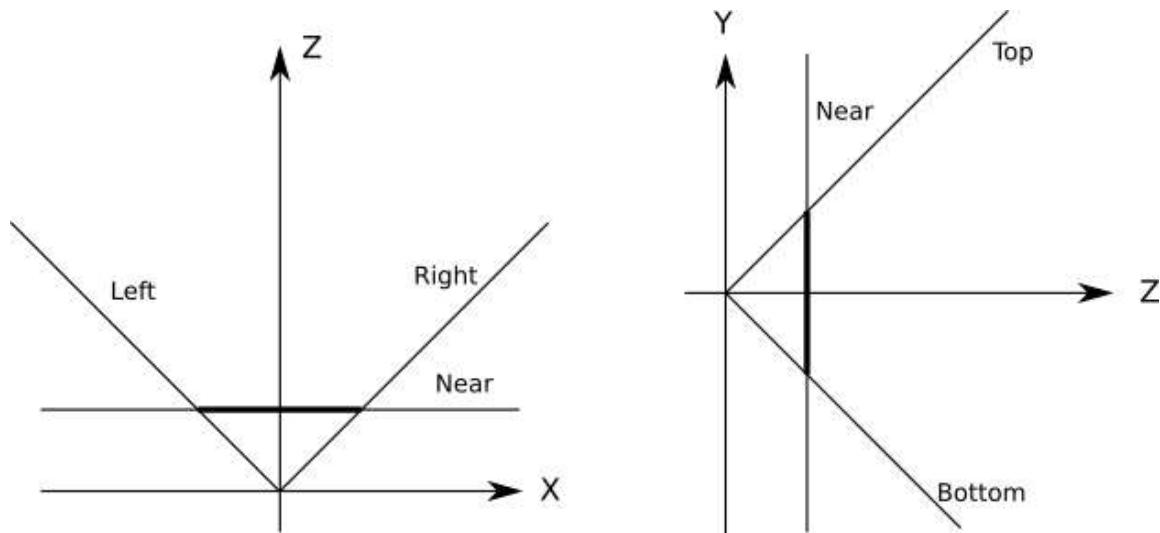


图 11-2：定义剪切体积的五个平面

每个修剪平面将空间分成两部分，我们称之为半空间。“内部”半空间是飞机前方的一切；“外部”半空间是它背后的一切。我们正在定义的剪切体积的“内部”是每个修剪平面定义的“内部”半空间的交点。在这种情况下，剪切体积看起来像一个无限高的金字塔，顶部被砍掉了。

这意味着，要针对剪切体积裁剪场景，我们只需要针对定义剪切体积的每个平面连续修剪它。修剪一个平面后剩余的任何几何图形都会被修剪到其余平面上。针对所有平面裁剪场景后，剩余的几何体是针对剪切体积裁剪场景的结果。

接下来，我们将了解如何针对每个剪切平面裁剪场景。

针对平面剪切场景

考虑一个包含多个对象的场景，每个对象由四个三角形组成（图 11-3）。

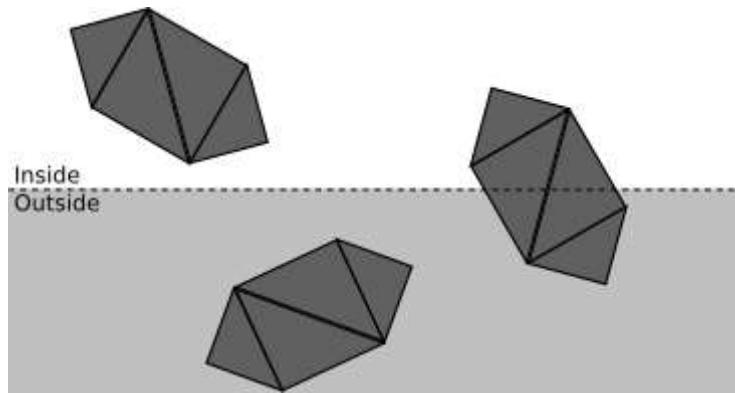


图 11-3：包含三个对象的场景

我们执行的操作越少，渲染器的速度就越快。我们将在剪裁平面上剪辑场景作为一系列阶段。每个阶段将尝试对尽可能多的几何图形进行分类，无论是接受还是放弃，具体取决于它是在修剪平面定义的半空间（即此平面的裁剪体积）内部还是之外。任何无法分类的几何图形都会进入下一阶段，这将更详细地研究它。

第一阶段尝试一次对整个对象进行分类。如果对象完全在剪切体积内，则接受该对象（图 11-4 中的绿色）；如果它完全在外面，则将其丢弃（图 11-4 中的红色）。

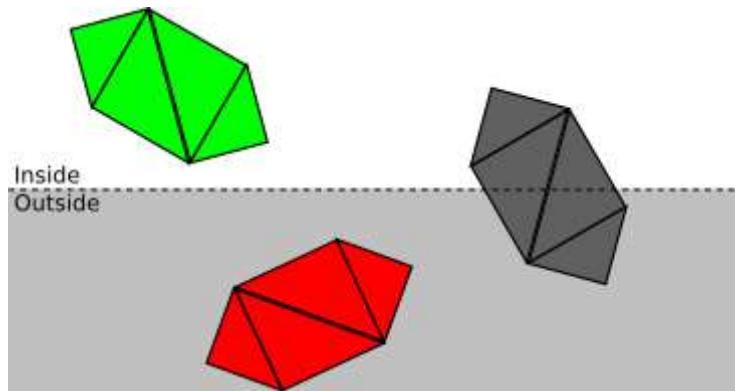


图 11-4：对象级别的剪裁。接受绿色，丢弃红色，灰色需要进一步处理。

如果一个对象不能被完全接受或丢弃，我们进入下一阶段，并独立地对它的每个三角形进行分类。如果三角形完全在剪切体积内，则接受；如果它完全在外部，则将其丢弃（请参阅图 11-5）。

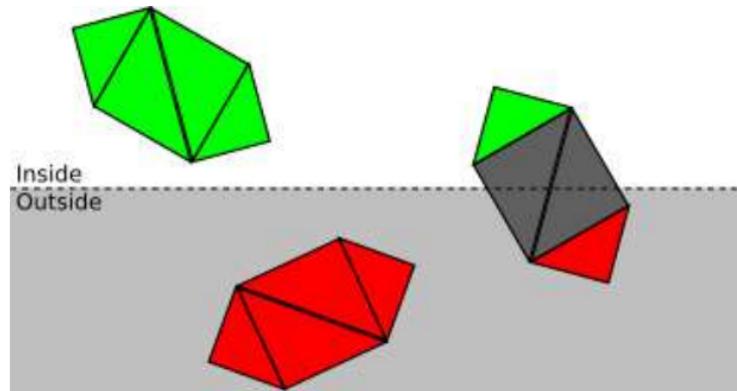


图 11-5：三角形级别的剪切。最右边对象的每个三角形要么被接受，要么被丢弃，要么需要进一步处理。

最后，对于每个未被接受或丢弃的三角形，我们需要裁剪三角形本身。删除原始三角形，并添加一个或两个新三角形以覆盖剪切体积内的三角形部分（请参阅图 11-6）。

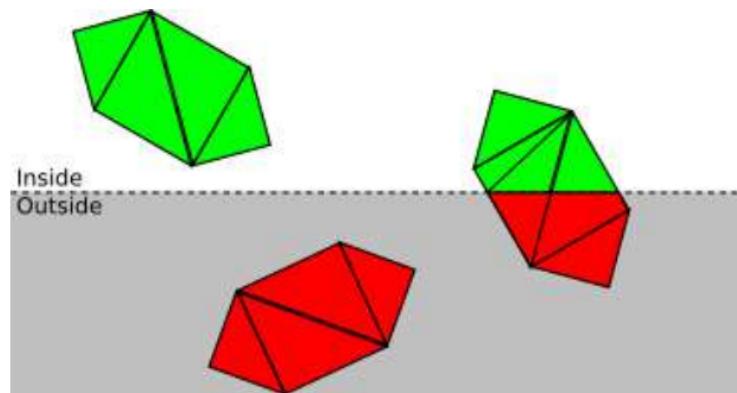


图 11-6：顶点级别的裁剪。部分位于剪辑卷内的每个三角形将拆分为一个或两个完全位于剪辑卷内的三角形。

现在我们对裁剪的工作原理有了清晰的概念理解，我们将开发数学和算法来创建有效的实现。

定义剪裁平面

让我们从投影平面 $Z = d$ 的方程开始，我们将它用作裁剪平面。这个等式很容易可视化，但对于我们的目的来说，它并不是最方便或最通用的形式。

3D 平面的一般方程是 $Ax + By + Cz + D = 0$ ，这意味着点 $P = (x, y, z)$ 将满足该方程当且仅当 P 在平面上。如果我们将系数 (A, B, C) 分组到一个向量 \vec{N} 中，我们可以将方程重写为 $\langle \vec{N}, P \rangle + D = 0$ 。

请注意，如果 $\langle \vec{N}, P \rangle + D = 0$ 则 $k\langle \vec{N}, P \rangle + kD = 0$ 对于 k 的任何值。特别是，我们可以选择 $k = 1/|\vec{N}|$ ，将原始方程相乘，得到一个新方程 $\langle \vec{N}', P \rangle + D' = 0$ ，其中 \vec{N}' 是一个单位矢量。因此，任何给定的平面都可以用方程 $\langle \vec{N}, P \rangle + D = 0$ 表示，其中 \vec{N} 是单位向量， D 是实数。

这是一个非常方便的公式： \vec{N} 恰好是平面的法线， $-D$ 是从原点到平面的有符号距离。事实上，对于任何点 P ， $\langle N, P \rangle + D$ 是从平面到 P 的有符号距离；距离 = 0 只是平面中包含 P 的特殊情况。

如果 \vec{N} 是平面的法线， $-\vec{N}$ 也是如此，所以我们选择 \vec{N} 使其指向剪切体积的“内部”。对于平面 $Z = d$ ，我们选择法线 $(0, 0, 1)$ ，它相对于相机指向“前进”。由于点 $(0, 0, d)$ 包含在平面中，它必须满足平面方程，我们可以求解 D ：

$$\langle \vec{N}, P \rangle + D = \langle (0, 0, 1), (0, 0, d) \rangle + D = d + D = 0$$

由此我们立即得到 $D = -d$ 。

我们可以直接从原始平面方程 $Z = d$ 中得到 $D = -d$ ，方法是将其重写为 $Z - d = 0$ 。但是，我们可以应用这种通用方法来推导出其余修剪平面的方程。

我们知道所有这些附加平面都有 $D = 0$ （因为它们都穿过原点），所以我们需要做的就是确定它们的法线。为了使数学简单，我们将选择一个 90° 视野(FOV)，这意味着平面位于 45° 。

考虑左修剪平面。其法线方向为 $(1, 0, 1)$ （即 45° 向右和向前）。该向量的长度是 $\sqrt{2}$ ，所以如果我们规范化它，我们会得到 $(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$ ，因此，左裁剪平面的方程式为

$$\langle N, P \rangle + D = \left\langle \left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right), P \right\rangle = 0$$

同样，右、下和上修剪平面的法线分别为 $(\frac{-1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$ 、 $(0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ 和 $(0, \frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ 。计算任意 FOV 的剪裁平面只需要一点点三角函数。

总之，我们的剪切体积由以下五个平面定义：

$$(near) \langle (0, 0, 1), P \rangle - d = 0 \quad (right) \langle \left(\frac{-1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right), P \rangle = 0 \quad (top) \langle (0, \frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}), P \rangle = 0 \\ (left) \langle \left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right), P \rangle = 0 \quad (bottom) \langle (0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}), P \rangle = 0$$

现在让我们详细了解如何针对平面裁剪几何图形。

剪切整个对象

假设我们将每个模型放在可以包含它的最小球体中；我们称该球体为物体的边界球体。计算这个球体比看起来要困难得多，它超出了本书的范围。但是，可以通过首先计算球体中心来获得粗略的近似值，方法是平均模型中所有顶点的坐标，然后将半径定义为从中心到最远的顶点的距离。

无论如何，让我们假设我们知道完全包含每个模型的球体的中心 C 和半径 r 。图 11-7 显示了一个包含一些对象及其边界球体的场景。

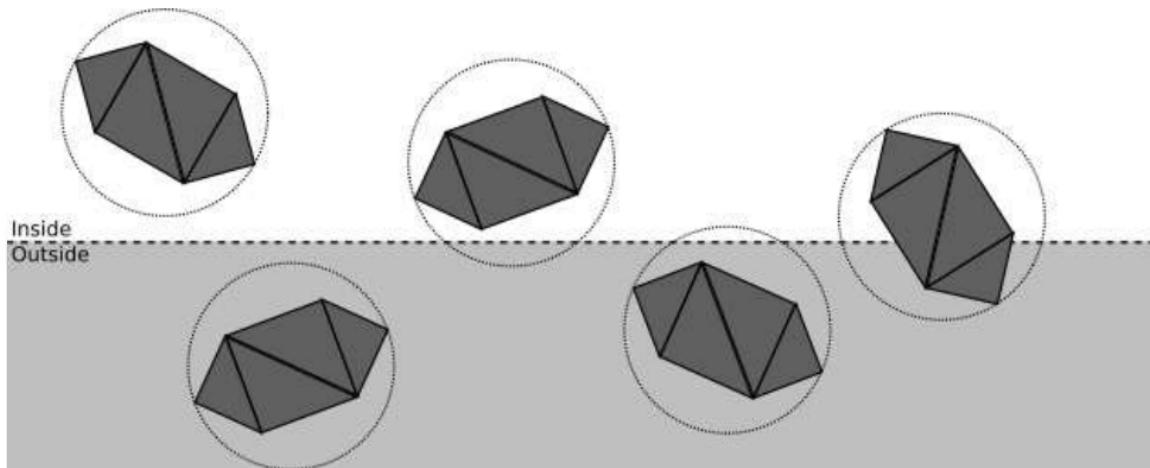


图 11-7：包含几个对象及其边界球体的场景

我们可以将这个球体和一个平面之间的空间关系分类如下：

球体完全在平面前方。

在这种情况下，接受整个对象；不需要对此平面进行进一步的修剪（但仍可能由其他平面进行修剪）。有关示例，请参见图 11-8。

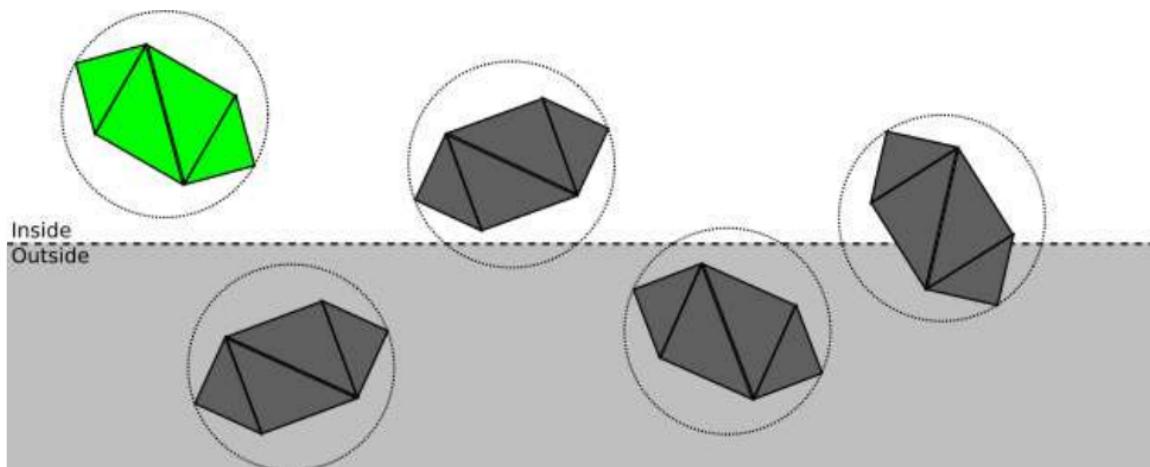


图 11-8：接受绿色对象。

球体完全在平面后面。

在这种情况下，将丢弃整个对象；无需进一步剪切（无论其他平面是什么，对象的任何部分都不会在剪切体积内）。有关示例，请参见图 11-9。

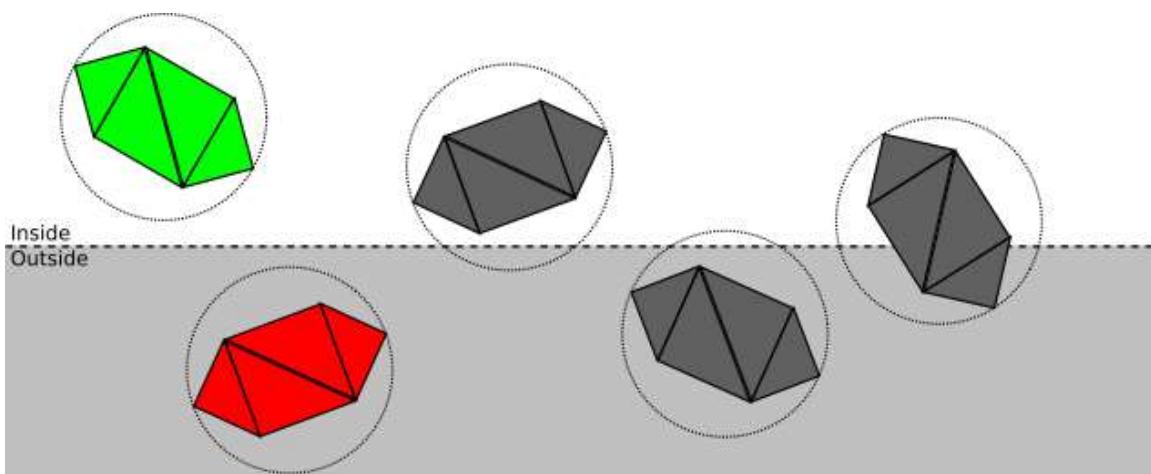


图 11-9：丢弃红色对象。

平面与球体相交。

这没有为我们提供足够的信息来了解对象的任何部分是否在剪切体积内;它可能完全在里面, 完全在外面, 或者部分在里面。有必要继续下一步并逐个三角形裁剪模型三角形。有关示例, 请参见图 11-10。

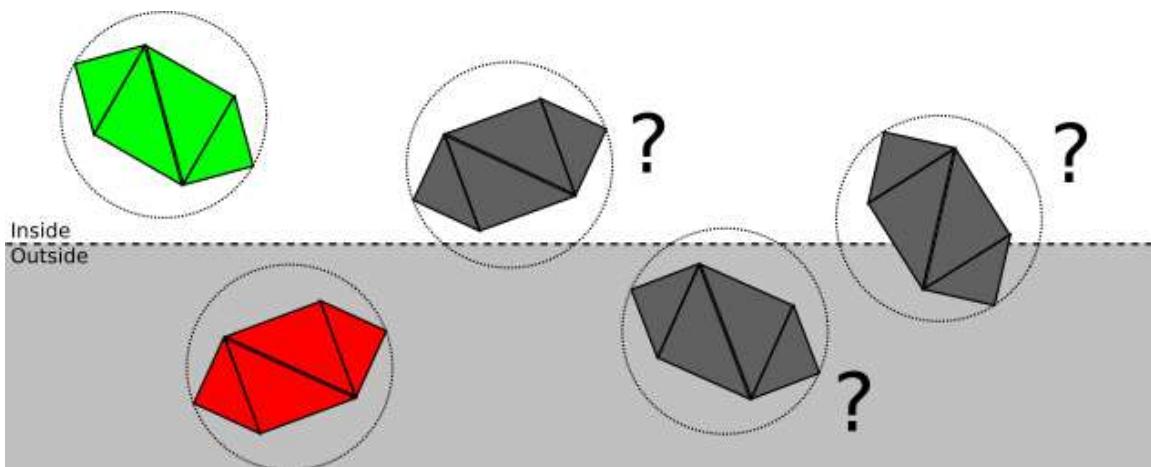


图 11-10：无法完全接受或丢弃灰色对象。

这种分类实际上是如何运作的? 我们选择表达剪裁平面的方式是, 将任何点插入平面方程中都会得到从点到平面的有符号距离;特别是, 我们可以计算有符号距离 d 从边界球体的中心到平面。所以如果 $d > r$, 球体在平面前方;如果 $d < -r$, 球体在平面后面;否则 $|d| < r$, 这意味着平面与球体相交。图 11-11 说明了所有三种情况。

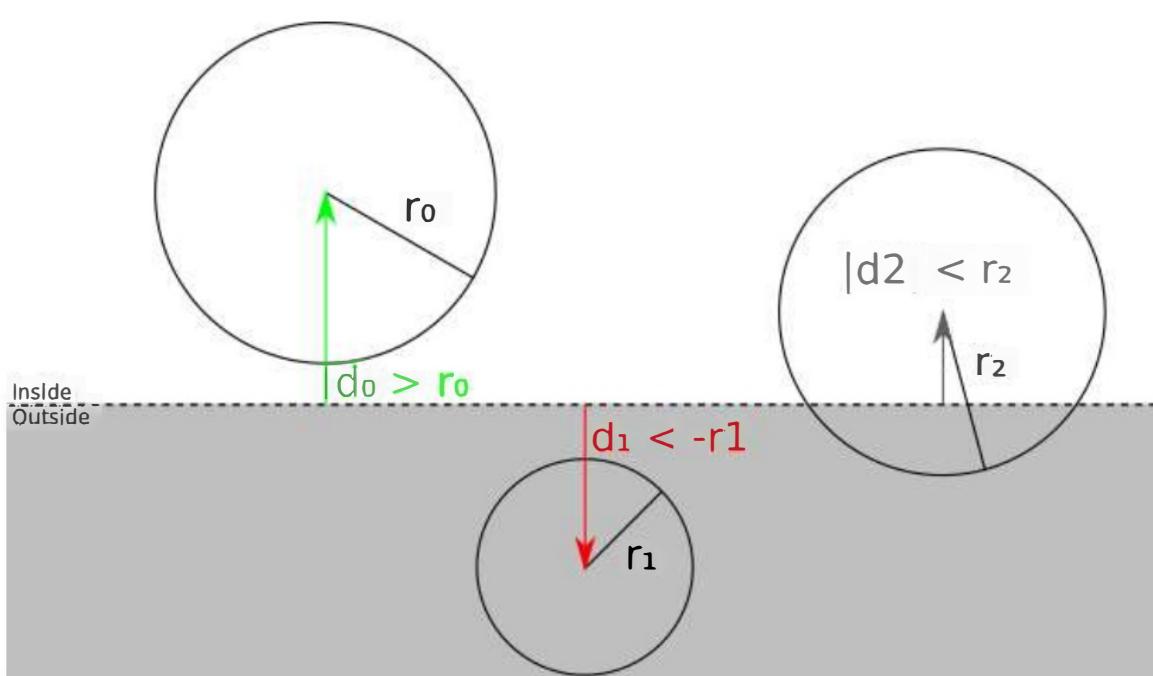


图 11-11：从球体中心到修剪平面的符号距离告诉我们球体是在平面前面、平面后面还是与平面相交。

剪裁三角形

如果球面-平面测试不足以确定对象是完全在剪切平面的前面还是完全在剪切平面的后面，我们必须对每个三角形进行裁剪。

我们可以通过查看三角形到平面的符号距离来对三角形的每个顶点进行分类。如果距离为零或正，则顶点位于修剪平面的前面；否则，它落后了。图 11-12 说明了这个想法。

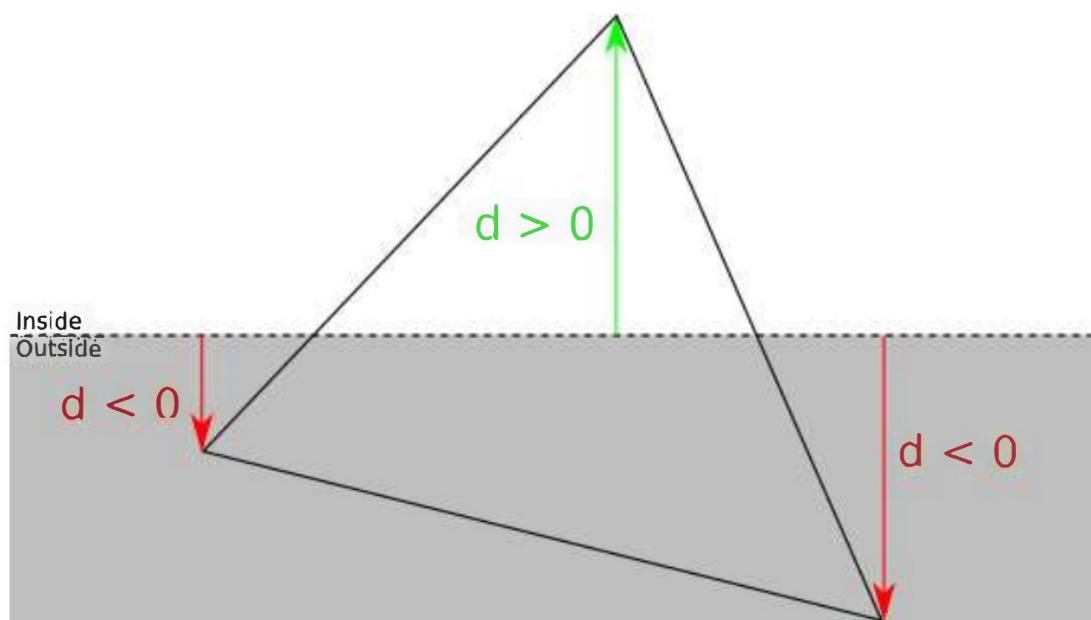


图 11-12：从顶点到修剪平面的有符号距离告诉我们顶点是在平面前面还是后面。

对于每个三角形，有四种可能的分类：

前面有三个顶点。

在这种情况下，整个三角形位于修剪平面的前面，因此我们接受它，并且不需要对该平面进行进一步的裁剪。

后面有三个顶点。

在这种情况下，整个三角形位于修剪平面后面，因此我们丢弃它，根本不需要进一步的裁剪。

前面有一个顶点。

设A是平面前面的三角形ABC的顶点。在这种情况下，我们丢弃ABC，并添加一个新的三角形AB'C'，其中B'和C'是AB和AC的交点与剪裁平面（图11-13）。

前面有两个顶点。

设A和B是三角形ABC在平面前面的顶点。在这种情况下，我们放弃ABC，添加两个新的三角形：ABA'和A'BB'，其中A'和B'是AC和BC与剪裁平面的交点（图11-14）。

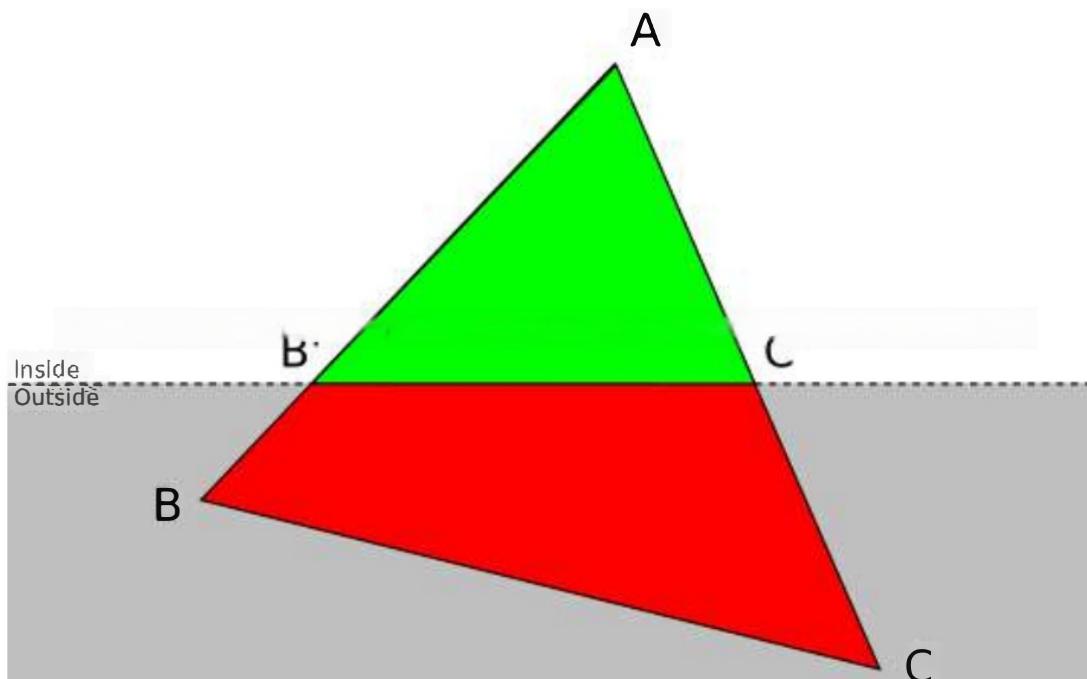


图11-13：剪切体积内部有一个顶点，外部有两个顶点的三角形ABC被单个三角形AB'C'代替。

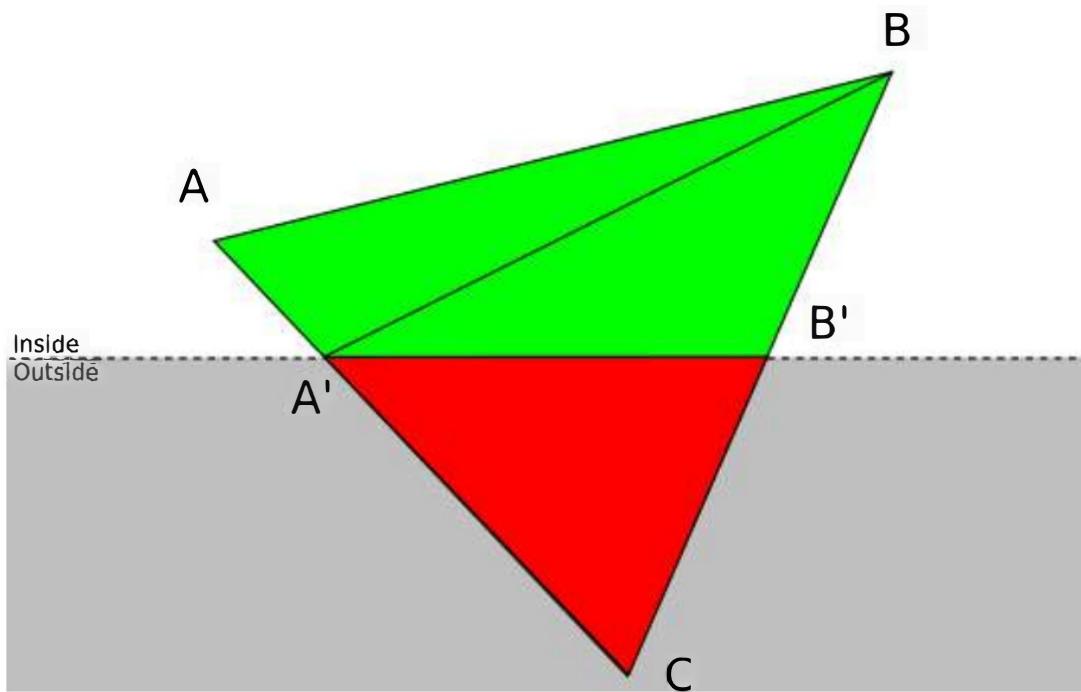


图 11-14: 剪裁体积外有一个顶点, 内部有两个顶点的三角形 ABC 被两个三角形 ABA' 和 $A'BB'$ 替换。

线段-平面相交

要像上面讨论的那样裁剪三角形, 我们需要计算三角形边与裁剪平面的交点。

我们有一个由方程给出的剪裁平面 $\langle N, P \rangle + D = 0$. 三角形边 $A B$ 可以用参数方程表示为 $P = A + t (B - A)$ 为 $0 \leq t \leq 1$. 计算参数的值 t 在发生交叉点的地方, 我们替换 P 在带有线段参数方程的平面方程中:

$$\begin{aligned} \langle N, P \rangle + D &= 0 \\ P = A + t (B - A) & \\ \Rightarrow \langle N, A + t (B - A) \rangle + D &= 0 \end{aligned}$$

使用点积的线性属性:

$$\langle N, A \rangle + t \langle N, B - A \rangle + D = 0$$

求解 t :

$$t = \frac{-D - \langle N, A \rangle}{\langle N, B - A \rangle}$$

我们知道解决方案总是存在的, 因为我们知道 $A B$ 与平面相交; 数学 $\langle N, B - A \rangle$ 不能为零, 因为这意味着线段和法线是垂直的, 这反过来又意味着线段和平面不相交。

已计算 t 、交叉路口 Q 只是

$$Q = A + t (B - A)$$

请注意，如果原始顶点带有其他属性（例如， h 我们在[第 7 章（填充三角形）](#)中使用的强度值），我们需要计算新顶点的这些属性的值。

在上面的等式中， t 是段的分数 A B 交叉点发生的地方。让 α_A 和 α_B 是某个属性在点 A 和 B 的值；如果我们假设属性在 A B 上线性变化，那么 α_Q 可以计算为

$$\alpha_Q = \alpha_A + t(\alpha_B - \alpha_A)$$

我们现在拥有了实现剪辑管道的所有算法和方程式。

剪切伪代码

让我们为剪辑管道编写一些高级伪代码。我们将遵循之前开发的自上而下的方法。

为了裁剪场景，我们剪辑了它的每个实例（示例 11-1）。

```
ClipScene(scene, planes) {
    clipped_instances = []
    for I in scene.instances {
        clipped_instance = ClipInstance(I, planes)
        if clipped_instance != NULL {
            clipped_instances.append(clipped_instance)
        }
    }
    clipped_scene = Copy(scene)
    clipped_scene.instances = clipped_instances
    return clipped_scene
}
```

示例 11-1：一种针对一组剪裁平面裁剪场景的算法

要裁剪一个实例，我们要么接受它，要么拒绝它，要么裁剪它的每一个三角形，这取决于它的边界球体（示例 11-2）。

```
ClipInstance(instance, planes) {
    for P in planes {
        instance = ClipInstanceAgainstPlane(instance, plane)
        if instance == NULL {
            return NULL
        }
    }
    return instance
}

ClipInstanceAgainstPlane(instance, plane) {
    d = SignedDistance(plane, instance.bounding_sphere.center)
    if d > r {
        return instance
    } else if d < -r {
        return NULL
    } else {
        clipped_instance = Copy(instance)
        clipped_instance.triangles =
            ClipTrianglesAgainstPlane(instance.triangles, plane)

        return clipped_instance
    }
}
```

示例 11-2：一种针对一组修剪平面裁剪实例的算法

最后，要裁剪一个三角形，我们要么接受它，要么拒绝它，要么将其分解为最多两个三角形，这取决于它在修剪平面前面有多少个顶点（示例 11-3）。

```
ClipTrianglesAgainstPlane(triangles, plane) {
    clipped_triangles = []
    for T in triangles {
        clipped_triangles.append(ClipTriangle(T, plane))
    }
    return clipped_triangles
}

ClipTriangle(triangle, plane) {
    d0 = SignedDistance(plane, triangle.v0)
    d1 = SignedDistance(plane, triangle.v1)
    d2 = SignedDistance(plane, triangle.v2)

    if {d0, d1, d2} are all positive {
        return [triangle]
    } else if {d0, d1, d2} are all negative {
        return []
    } else if only one of {d0, d1, d2} is positive {
        let A be the vertex with a positive distance
        compute B' = Intersection(AB, plane)
        compute C' = Intersection(AC, plane)
        return [Triangle(A, B', C')]
    } else /* only one of {d0, d1, d2} is negative */ {
        let C be the vertex with a negative distance
        compute A' = Intersection(AC, plane)
        compute B' = Intersection(BC, plane)
        return [Triangle(A, B, A'), Triangle(A', B, B')]
    }
}
```

示例 11-3：一种针对修剪平面裁剪一组三角形的算法

辅助函数只是将点的坐标代入平面方程中（示例 11-4）。SignedDistance

```
SignedDistance(plane, vertex) {
    normal = plane.normal
    return (vertex.x * normal.x)
        + (vertex.y * normal.y)
        + (vertex.z * normal.z)
        + plane.D
}
```

示例 11-4：计算从平面到点的有符号距离的函数

[源代码和现场演示>>](#)

渲染管线中的剪辑

书中章节的顺序不是渲染管线中操作的顺序；正如导言中所解释的，各章的排序方式是尽快取得可见的进展。

剪切是一种 3D 操作；它在场景中获取 3D 对象并在场景中生成一组新的 3D 对象，或者更准确地说，它计算场景和剪切体积的交集。因此，必须在将对象放置在场景中之后（即，在模型和照相机变换后使用顶点）但在透视投影之前进行裁剪。

本章介绍的技术工作可靠，但非常通用。您对场景的先验知识越多，剪辑的效率就越高。例如，许多游戏通过向关卡添加可见性信息来预处理其关卡；如果可以将场景划分为“房间”，则可以制作一个表，列出从任何给定房间可见的房间。稍后渲染场景

时，您只需要弄清楚摄像机在哪个房间，就可以安全地忽略所有标记为“不可见”的房间，从而在渲染过程中节省大量资源。当然，权衡是更多的预处理时间和更僵化的场景。如果对本主题感兴趣，请阅读 BSP 分区和门户系统。

总结

在本章中，我们终于解除了透视投影方程造成的主要限制之一。我们已经克服了只有相机前方的顶点才能有意义地投影的限制。为了做到这一点，我们对“在镜头前”的含义提出了一个精确的定义：无论在剪辑体积内是什么，我们用五个平面来定义。

然后，我们开发了方程和算法来计算场景和剪切体积之间的几何交集。因此，我们可以拍摄整个场景并删除所有不可能投影到视口上的内容。这不仅避免了透视投影方程无法处理的情况，而且还通过删除将在视口之外投影的几何体来节省计算资源。

但是，在剪裁场景后，我们可能最终会得到在最终画布中可见的几何体，但不会，很可能是因为前面还有其他东西！我们将在下一章中找到处理此问题的方法。