# 距摄像机一定距离的视锥体的大小

距摄像机一定距离的视锥体的横截面将在世界空间中定义一个构成可见区域的矩形。此形状有时可用于计算此矩形在给定距离处的大小，或者查找矩形为给定大小时所处的距离。例如，如果移动的摄像机需要始终将对象（例如玩家）完全保持在镜头中，则不得过于接近摄像机以免该对象的一部分被截断。

可使用以下公式来计算视锥体在给定距离处的高度（均以世界单位表示）：

var frustumHeight = 2.0f \* distance \* Mathf.Tan(camera.fieldOfView \* 0.5f \* Mathf.Deg2Rad);

…还可反转该过程以计算获得指定视锥体高度所需的距离：

var distance = frustumHeight \* 0.5f / Mathf.Tan(camera.fieldOfView \* 0.5f \* Mathf.Deg2Rad);

当高度和距离已知时，也可以计算 FOV（视野）角度：

var camera.fieldOfView = 2.0f \* Mathf.Atan(frustumHeight \* 0.5f / distance) \* Mathf.Rad2Deg;

这些计算都涉及到视锥体的高度，但这可以很容易从宽度（反之亦然）获得：

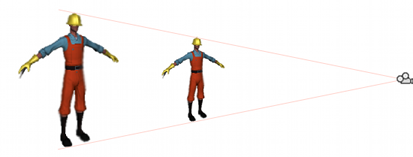
var frustumWidth = frustumHeight \* camera.aspect;

var frustumHeight = frustumWidth / camera.aspect;

# 推拉变焦（也称为“伸缩变焦”效果）

推拉变焦 (Dolly Zoom) 是众所周知的摄像机同时向目标对象移动并将其缩小的视觉效果。结果是对象看起来大小大致相同，但场景中的其他所有对象都改变了视角。巧妙进行推拉变焦可实现突出目标对象的效果，因为它是场景中不会在图像中移位的唯一对象。或者，可特意快速执行变焦以产生迷失方向的效果。

刚好垂直装入视锥体内的对象将占据屏幕上看到的整个视图的高度。无论对象与摄像机的距离以及视野如何，都是如此。例如，您可以将摄像机移近对象，但随后加宽视野，使对象仍然刚好装在视锥体的高度内。该特定对象在屏幕上将显示相同的大小，但随着距离和 FOV（视野）的变化，其他所有对象都将改变大小。这就是推拉变焦效果的本质。



通过代码创建该效果实际上就是在变焦开始时将视锥体高度保存在对象的位置。然后，摄像机移动时，找到它的新距离并调整 FOV 以使其在对象位置保持相同高度。通过以下代码可实现这一点：



# 摄像机射线

[了解视锥体](file:///E:\UnityDocumentation\Manual\UnderstandingFrustum.html)部分说明了摄像机视图中的任何一点都对应于世界空间中的一条线。有时使用这条线的数学表示形式是有用的，Unity 能够以 [Ray](file:///E:\UnityDocumentation\ScriptReference\Ray.html) 对象的形式提供该表示形式。Ray 始终对应于视图中的一个点，因此 Camera 类提供 [ScreenPointToRay](file:///E:\UnityDocumentation\ScriptReference\Camera.ScreenPointToRay.html) 和 [ViewportPointToRay](file:///E:\UnityDocumentation\ScriptReference\Camera.ViewportPointToRay.html) 函数。两者之间的区别在于 ScreenPointToRay 期望以像素坐标的形式提供该点，而 ViewportPointToRay 则接受 0..1 范围内的标准化坐标（其中 0 表示视图的左下角，1 表示右上角）。这些函数中的每一个函数都返回由一个原点和一个矢量（该矢量显示从该原点出发的线条方向）组成的 Ray。射线 (Ray) 源自近裁剪面而不是摄像机 (Camera) 的 transform.position 点。

## 射线投射

来自摄像机的射线最常见的用途是将[射线投射 (raycast)](file:///E:\\UnityDocumentation\\ScriptReference\\Physics.Raycast.html) 到场景中。射线投射从原点沿着射线方向发送假想的“激光束”，直至命中场景中的碰撞体。随后会返回有关该对象和 [RaycastHit](file:///E:\UnityDocumentation\ScriptReference\RaycastHit.html) 对象内的投射命中点的信息。这是一种基于对象在屏幕上的图像来定位对象的非常有用的方法。例如，可使用以下代码确定鼠标位置处的对象：

# 使用斜视锥体

默认情况下，视锥体围绕摄像机的中心线对称安放，但这并不是必须的。视锥体可设置为“倾斜的”，即一侧与中心线的角度小于对侧与中心线的角度。效果就像拍摄打印的照片并切掉一条边。这种做法使得图像一侧的透视看起来更加紧凑，给人的印象是观察者非常靠近在该边缘处可见的对象。此功能的一个用法示例是赛车游戏，这种情况下的视锥体可能在其底边处变得扁平。因此，观察者看起来会更接近路面，突出了速度感。



虽然 Camera 类没有设置视锥体倾斜度的函数，但通过改变投影矩阵可以很容易实现这一点：

幸运的是，没有必要了解投影矩阵如何使用它。horizObl 和 vertObl 值分别设置水平和垂直倾斜量。值为零表示无倾斜。正值使视锥体向右或向上移动，从而使左边或底边变平。负值使视锥体向左或向下移动，从而使视锥体的右边或顶边变平。如果将此脚本添加到摄像机并在游戏运行时将游戏切换到 Scene 视图，则可以直接看到效果；在检视面板中改变 horizObl 和 vertObl 的值时，摄像机视锥体的线框会发生变化。任一变量中的值为 1 或 –1 表示视锥体的一侧与中心线完全齐平。此范围之外的值也是允许使用的，但通常没有必要

# 创建尺寸大小的效果

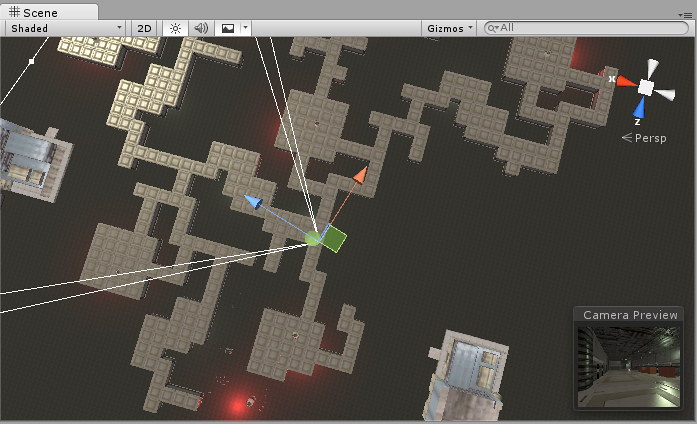
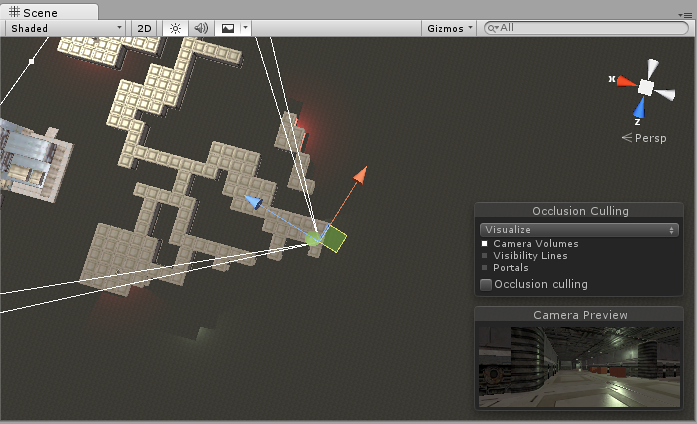
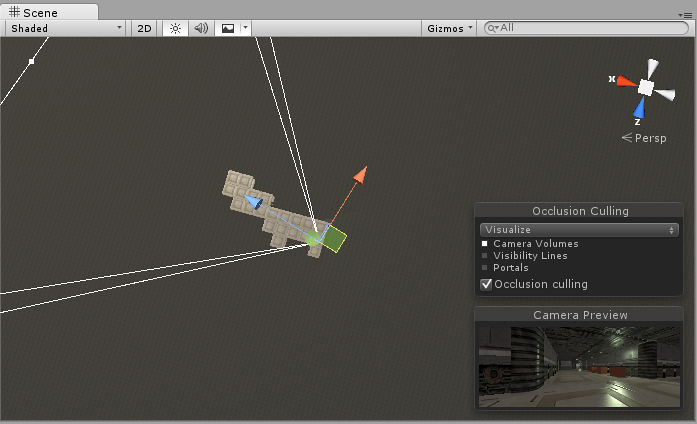
从图形的角度来看，Unity 中的距离单位是任意的，与现实世界的测量值不对应。虽然这为设计提供了灵活性和便利性，但是传达对象的预期尺寸并不是那么容易。例如，玩具车即便可能是真实物体的精确比例模型，看上去也会与真实尺寸车辆有所不同。

影响对象给人的大小尺寸印象的一个重要元素是透视角度随着对象长度的变化情况。例如，如果从后面观看玩具车，那么玩具车前面仅比后面的距离远一点点。由于距离不远，透视角度产生的影响相对较小，因此在车前面与车后面的尺寸比例相差很小。但如果是正常尺寸的汽车，那么车前部与车后部之间与摄像机的距离将相差几米，因此透视角度的影响将更加明显。

要让对象看上去很小，透视线仅应该在其深度上略微偏离。为此，您可以使用比默认值 60 度更窄的视野，并将摄像机移动得更远以补偿增加的屏幕尺寸。相反，如果希望让对象看起来很大，请使用宽视野 (FOV) 并拉近摄像机。当这些透视变换技巧与其他明显的技巧（比如从高于正常的有利位置俯视“小”物体）一起使用时，得到的结果会非常逼真。

# 遮挡剔除

当对象被其他对象阻挡（遮挡）而不能被摄像机所看到时，遮挡剔除 (Occlusion Culling) 功能会禁用对象的渲染。这种情况不会自动发生在 3D 计算机图形中，因为在大多数时间，距离摄像机最远的对象都是先绘制的，而较近的对象则在先前对象的基础上绘制（这称为“过度绘制 (Overdraw)”）。遮挡剔除与视锥体剔除 (Frustum Culling) 不同。视锥体剔除仅禁用摄像机视野之外的对象的渲染器，而不会禁用由过度绘制隐藏起来的任何对象的渲染器。请注意，使用遮挡剔除时，仍然会受益于视锥体剔除。

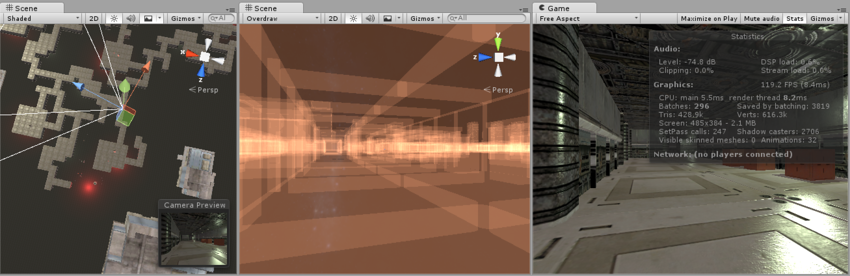
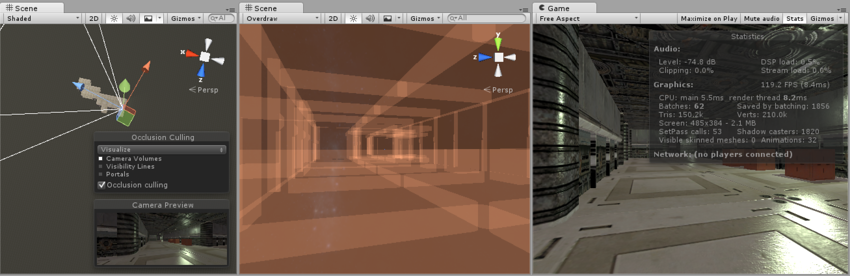
迷宫般的室内关卡。此常规 Scene 视图显示了所有可见的游戏对象。 常规视锥体剔除仅渲染摄像机视野内的对象。这是自动始终执行的原则。 如果有对象被更近的对象完全遮挡，则遮挡剔除可进一步从摄像机渲染工作中移除这些被遮挡的对象。

遮挡剔除过程将使用虚拟摄像机在场景中移动，进而构建潜在可见对象集的层级视图。每个摄像机在运行时都会使用此数据来识别可见和不可见的对象。凭借此信息，Unity 将确保只发送可见对象进行渲染。这样可减少绘制调用次数并提高游戏性能。

遮挡剔除的数据由单元格组成。每个单元格是从整个场景包围体上细分而来。具体地来说，这些单元格形成一个二叉树。遮挡剔除使用两个树，一个用于视图单元格（静态对象），另一个用于目标单元格（移动对象）。视图单元格映射到一个定义可见静态对象的索引列表，从而为静态对象提供更准确的剔除结果。

在创建对象时记住这一点很重要，因为您需要在对象大小和单元格大小之间取得良好的平衡。理想情况下，单元格与对象相比不应该太小，但同样，对象不应该覆盖大量单元格。有时，可通过将大型对象分成更小的碎片来改善剔除。但是，您仍然可以将小对象合并在一起以减少绘制调用，只要它们都属于同一个单元格，就不会影响遮挡剔除。

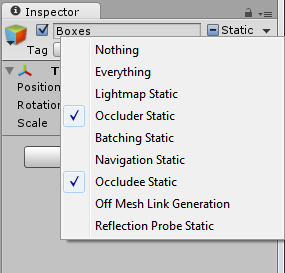
您可以使用“过度绘制”场景渲染模式来查看正在发生的过度绘制量，并可使用 Game 视图中的统计信息面板查看正在渲染的三角形、顶点和批次的数量。下面是应用遮挡剔除之前和之后的比较。

注意在过度绘制的 Scene 视图中，由于渲染了可见墙壁背后的很多房间，因此过度绘制的密度很高。这些对象在 Game 视图中不可见，但是仍然要用时间来渲染它们。 应用遮挡剔除后，不会渲染远处房间，过渡绘制的密度大幅降低，渲染的三角形和批次数量急剧下降，Game 视图的外观没有任何变化。

## 设置遮挡剔除

为了使用遮挡剔除，需要进行一些手动设置。首先，关卡几何体必须分成尺寸合理的碎片。将关卡布置到大型对象旁（例如墙壁、建筑物等）彼此遮挡且界限分明的小型区域中，也很有帮助。此处的思路是基于遮挡数据开启或关闭每个单独的网格。因此，如果您有一个包含房间所有家具的对象，则整套家具将全部剔除或全部不剔除。这种方法并不像使每件家具拥有自己的网格（因此每个家具都可以根据摄像机的视点单独剔除）那样合理。

您需要在 **Inspector** 中把纳入遮挡计算的所有场景对象标记为 **Occluder Static**。执行此操作最快的方法是多选要包含在遮挡计算中的对象，然后将它们标记为 **Occluder Static** 和 **Occludee Static**。

标记要进行遮挡的对象

何时应使用 **Occludee Static\_\_？不会产生遮挡的完全透明或半透明对象，以及不太可能遮挡其他对象的小型对象应标记为\_\_被遮挡物 (Occludee)** 而不是\_\_遮挡物 (Occluder)\_\_。这意味着，它们将被其他对象遮挡，但本身不会被视为遮挡物，这样将有助于减少计算。

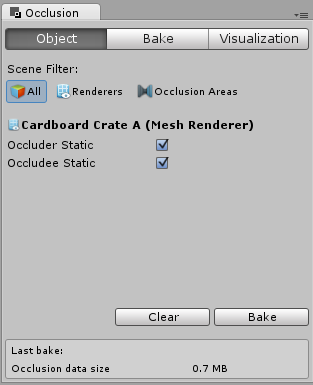
使用 LOD 组时，只有基本级别对象 (LOD0) 可用作遮挡物。

## 遮挡剔除窗口

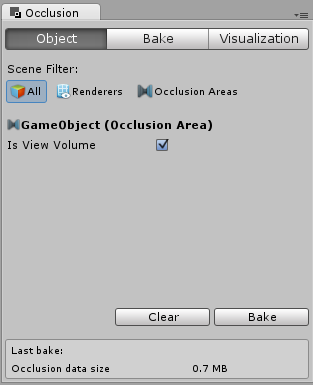
对于大多数处理遮挡剔除的操作，都应该使用遮挡剔除窗口 (**Window > Occlusion Culling**)

在遮挡剔除窗口中，可使用遮挡物网格和[遮挡区域 (Occlusion Areas)](file:///E:\\UnityDocumentation\\Manual\\class-OcclusionArea.html)。

如果您处于\_\_遮挡剔除窗口\_\_的 **Object** 选项卡中，并在场景中选择了[网格渲染器](file:///E:\\UnityDocumentation\\Manual\\class-MeshRenderer.html)，您就可以修改相关的静止标记：

网格渲染器的遮挡剔除窗口

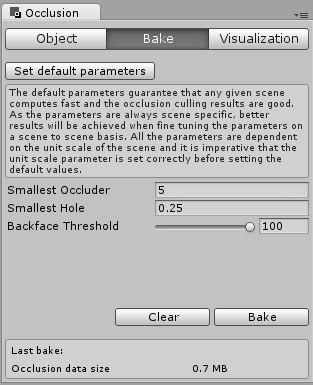
如果您处于\_\_遮挡剔除窗口\_\_的 **Object** 选项卡中，并选择了[遮挡区域 (Occlusion Area)](file:///E:\\UnityDocumentation\\Manual\\class-OcclusionArea.html)，您就可以使用相关的 OcclusionArea 属性（有关更多详细信息，请转到[遮挡区域](file:///E:\\UnityDocumentation\\Manual\\class-OcclusionArea.html)部分）

遮挡区域的遮挡剔除窗口

**注意：**默认情况下，如果不创建任何遮挡区域，遮挡剔除将应用于整个场景。

**注意：**只要摄像机位于遮挡区域之外，就不会应用遮挡剔除。设置遮挡区域来覆盖摄像机可能出现的位置非常重要的，但如果将这些区域设置得过大，烘焙过程中就会产生额外成本。

## 遮挡剔除 - Bake

遮挡剔除检视面板 Bake 选项卡。

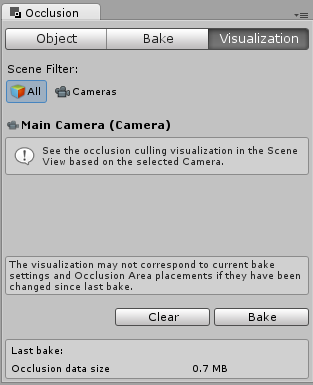
遮挡剔除烘焙窗口有一个“Set Default Parameters”按钮，允许您将烘焙值重置为 Unity 的默认值。这些值对于许多典型场景很有用，但您可以调整这些值来适应场景的特定内容，这通常可以产生更好的效果。

## 属性

| **属性** | **功能** |
| --- | --- |
| **Smallest Occluder** | 进行遮挡剔除时用于隐藏其他对象的最小尺寸。任何小于这个尺寸的对象都不会剔除它们所遮挡的对象。例如，当值为 5 时，高于或宽于 5 米的所有对象都将导致它们挡住的对象被剔除（不渲染，从而节省渲染时间）。为此属性选择一个合适的值，在遮挡数据的精度和存储大小之间取得平衡。 |
| **Smallest Hole** | 此值表示摄像机应该看到的几何体之间的最小间隙。此值表示可以穿过孔的对象的直径。如果场景中存在摄像机应该能够看到的小裂缝，则 Smallest Hole 值必须小于间隙的最窄尺寸。 |
| **Backface Threshold** | Unity 的遮挡使用数据大小优化功能，通过测试背面来减少不必要的细节。默认值 100 比较稳妥，绝不会从数据集删除背面。值 5 将基于具有可见背面的位置大幅减少数据。思路是，一般情况下，有效的摄像机位置通常不会看到太多背面；例如，地形底面的视图，或者您应该无法到达的实体对象内的视图。如果阈值低于 100，Unity 将完全从数据集删除这些区域，从而减少遮挡的数据大小。 |

Bake 选项卡的底部是 Clear 和 Bake 按钮。单击 **Bake** 按钮可开始生成遮挡剔除数据。生成数据后，可使用 Visualization 选项卡预览和测试遮挡剔除。如果对结果不满意，请单击 **Clear** 按钮删除以前计算的数据，调整设置，并再次烘焙。

## 遮挡剔除 - Visualization

遮挡剔除检视面板 Visualization 选项卡。

场景中的所有对象都会影响包围体的大小，因此请尽量将它们全部保留在场景的可见边界内。

准备好生成遮挡数据后，单击 **Bake** 按钮。请记住在 **Bake** 选项卡中选择 **Memory Limit**。较低的值可以加快生成速度，但会降低精确度，较高的值将生成更适合发行版的质量。

请记住，构建遮挡数据所需的时间取决于单元格级别、数据大小和您所选择的质量。

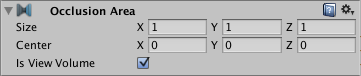
处理完成后，您应该可以在查看区域中看到一些彩色立方体。彩色区域是共享相同遮挡数据的区域。

如果要删除遮挡剔除的所有预计算数据，请单击 **Clear**。

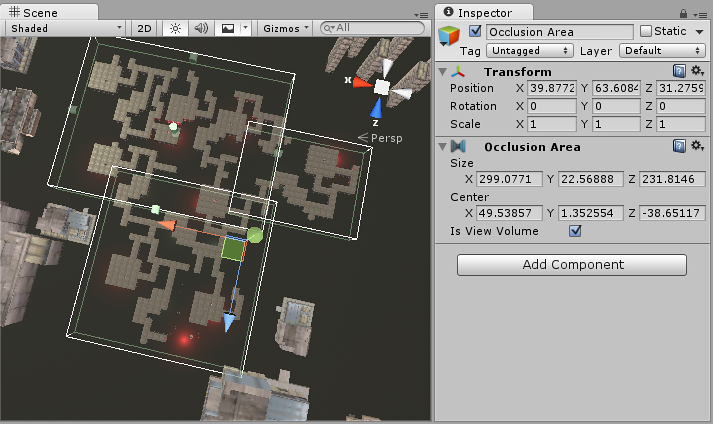
# 遮挡区域

要将遮挡剔除应用于移动对象，必须创建\_\_遮挡区域 (Occlusion Area)**，然后修改其大小以适合移动对象所在的空间（当然，移动对象不能标记为静态）。可通过将\_\_遮挡区域 (Occlusion Area)** 组件添加到空游戏对象来创建遮挡区域（菜单：\_\_Component > Rendering > Occlusion Area\_\_）。

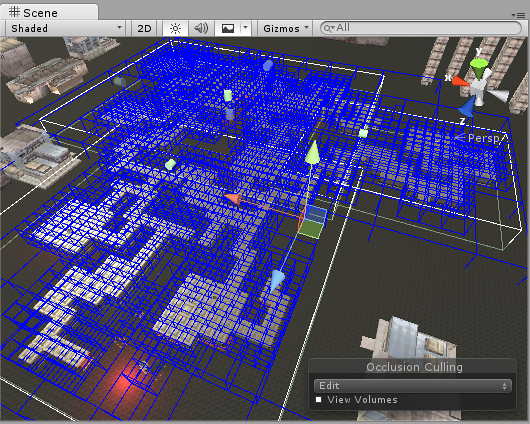
创建\_\_遮挡区域\_\_后，请选中 Is View Volume 复选框以遮挡移动对象。



| **属性：** | **功能：** |
| --- | --- |
| **Size** | 定义遮挡区域的大小。 |
| **Center** | 设置遮挡区域的中心。默认情况下，此设置为 0,0,0，位于盒体的中心。 |
| **Is View Volume** | 定义摄像机的位置。选中此项可遮挡该\_遮挡区域\_内的静态对象。 |

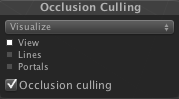
移动对象的遮挡区域 (Occlusion Area) 属性。

添加遮挡区域后，需要查看它如何将盒体分成单元格。要了解如何计算遮挡区域，请在 **Occlusion Culling 预览面板\_\_中 选择** Edit\_\_ 并切换 **View** 按钮。



## 测试生成的遮挡

设置遮挡后，可通过启用 Occlusion Culling\_（在 Visualize 模式下的 \_Occlusion Culling 预览面板\_\_中）并在 Scene 视图中移动\_\_主摄像机\_\_来测试遮挡。

Scene 视图中的 Occlusion View 模式

当您移动主摄像机时（无论是否处于播放模式下），您将看到各种对象自行禁用。此处要寻找的是遮挡数据是否有任何错误。如果在移动摄像机时看到有对象突然弹入视图，则表示有错误。如果发生这种情况，修复错误的方法为更改分辨率（如果正在使用目标体积），或者移动对象以掩盖错误。要调试遮挡问题，可将主摄像机移动到有问题的位置进行抽样检查。

处理完成后，您应该在查看区域中看到一些彩色立方体。蓝色立方体表示\_\_目标体积\_\_的单元格划分。白色立方体表示\_\_视图体积\_\_的单元格划分。如果参数设置正确，您应该看到一些对象未被渲染。这是因为它们位于摄像机的视锥体之外，或被其他对象遮挡。

遮挡完成后，如果在场景中没有看到任何对象被遮挡，请尝试将对象分成更小的部分，以便能将它们完全包含在单元格内。