

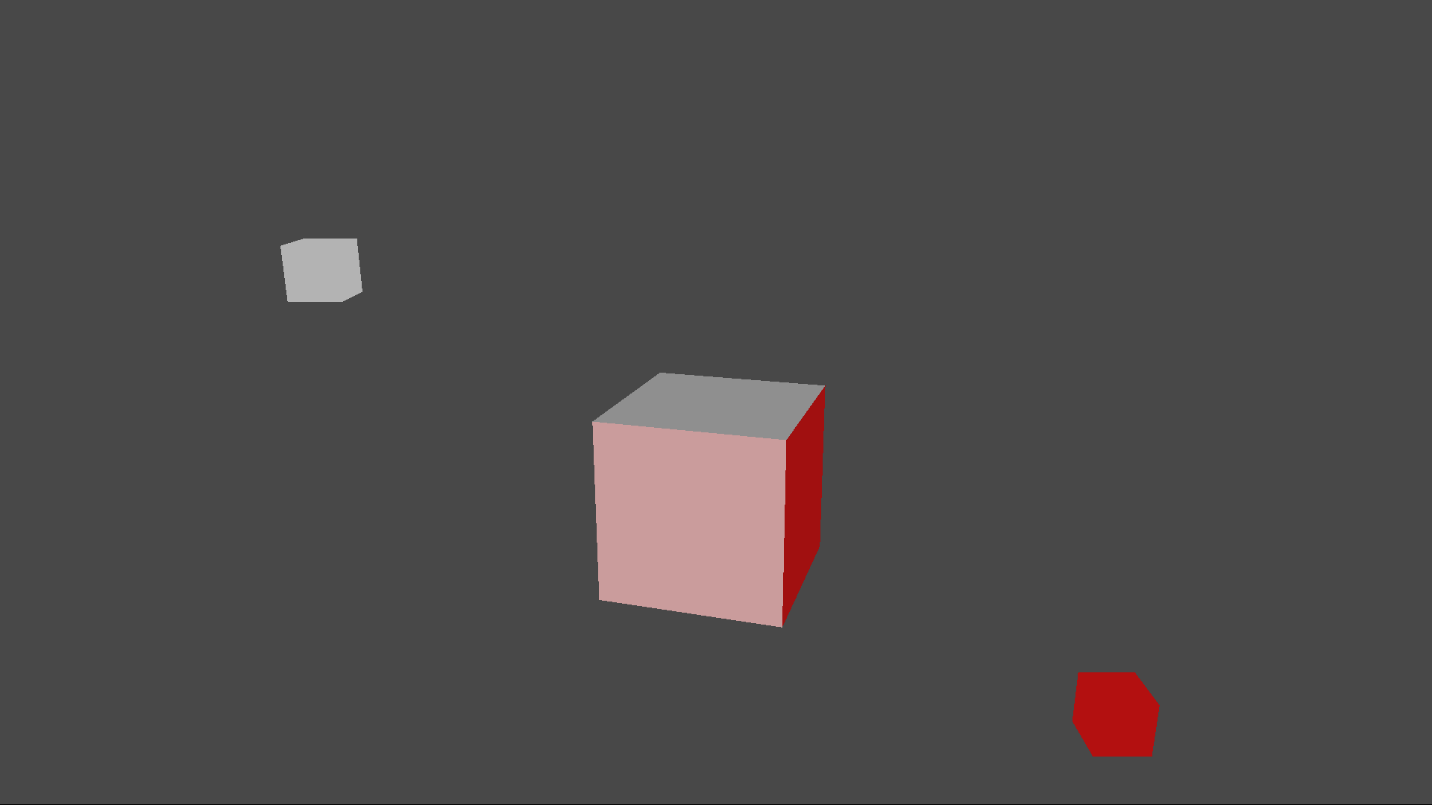
简单光照示例

此示例与 Microsoft 游戏开发工具包预览版（2019 年 11 月）兼容

# 说明

本示例演示如何创建静态 Direct3D 12 顶点、索引和常量缓冲区，以便使用静态和动态朗伯光照绘制已编制索引的几何照明。

本示例呈现一个大立方体，该立方体由两种光（一个白色光，一个红色光）照亮，这两个光源也以立方体的外形表示。白色光静止不动，而红色光围绕中心立方体旋转。中央立方体也会旋转。通过这个运动轨迹，你可以从不同角度观察彩色灯光的效果。



# 构建示例

如果使用 Xbox One 开发工具包，请将活动解决方案平台设置为 Gaming.Xbox.XboxOne.x64。

如果使用 Project Scarlett，请将活动解决方案平台设置为 Gaming.Xbox.Scarlett.x64。

有关详细信息，请参阅 GDK 文档中的“运行示例”。

# 使用示例

|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 退出 | “视图”按钮 |

# 实现说明

## 着色器

## 本示例使用三个着色器来呈现场景，分别是一个顶点着色器（“TriangleVS”）和两个像素着色器（“LambertPS”、“SolidColorPS”。） 在 CreateDeviceDependentResources 中加载已编译的着色器 blob，并在创建每个着色器组合的管道状态对象时引用它。 所有着色器都在同一个 HLSL 包含文件“SimpleLighting.hlsli”中定义，三个存根着色器都包含此文件。每个存根着色器都针对不同的入口点经过编译，用于创建三个着色器 blob。

## 管道状态对象 (PSO)

简单光照示例有两个唯一着色器组合：一个是 TriangleVS 与 LambertPS 的组合，另一个是 TriangleVS 与 SolidColorPS 的组合。在 DirectX 12 中，需要为每个唯一着色器组合创建管道状态对象 (PSO)。根据名称可以看出，PSO 使用一组特定的着色器封装你可能需要进行多次绘图调用的所有管道状态。PSO 将根签名、管道各个阶段的着色器、状态设置（如光栅器状态、深度模具状态、混合状态等）结合在一起（有关更多详细信息，请参阅 MSDN 上的文档。）

## 根签名

根签名定义绑定到图形管道的资源类型以及资源的布局方式。根签名类似于 API 函数签名，它描述参数类型、参数顺序和布局，但不定义任何实际参数实例。根参数是对应于根签名元素的实际数据实例。示例的顶点着色器只需要用于着色器常量的单一结构，因此根签名非常简单。根签名包含类型为 ConstantBufferView 的单个根参数。

## 几何图形

## 场景的几何图形由静态顶点和索引数组组成，该数组包含可表示立方体 6 个四边形的 24 个顶点的数据。这两个数组在 Sample::CreateDeviceDependentResources 内部声明，其中 ID3D12Device::CreateCommittedResource 立即使用这两个数组为缓冲区创建 ID3D12Resources。为简单起见，示例使用 D3D12\_HEAP\_TYPE\_UPLOAD，因为这样可以在单个步骤中创建每个资源并用数据初始化每个资源。不过，\_UPLOAD 堆是几何数据的次优位置。更有效的实现方式是将 D3D12\_HEAP\_TYPE\_DEFAULT 用于几何数据。为了初始化 \_DEFAULT 堆，需要使用 \_UPLOAD 堆，这样你最终会使用两个堆，这使实现变得复杂。

创建几何图形的缓冲区后，本示例可以为顶点创建一个 D3D12\_VERTEX\_BUFFER\_VIEW，并为索引创建一个 D3D12\_INDEX\_BUFFER\_VIEW。通过调用 ID3D12GraphicsCommandList::IASetVertextBuffer 和 ID3D12GraphicsCommandList::IASetVertextBuffer 来设置输入汇编程序时，在 Sample::Render 中使用视图。

## 管理着色器常量

在这个非常简单的场景中，所有着色器常量都集中在一个单一的常量缓冲区中，其中包含以下内容。

* 世界矩阵、视图矩阵和投影矩阵
* 光照方向和色彩
* 单色

对于更复杂的场景，你通常会根据常量的更新频率将这些常量拆分为多个缓冲区。

由于大立方体和红色光呈现动画效果，因此需要在两次绘制调用之间多次更新每帧的着色器常量。回顾一下，根签名包含类型为 ConstantBufferView 的单个根参数，这需要引用着色器常量的副本，以便在每个绘制调用中使用。由于 CPU 和 GPU 并行工作，因此，在 GPU 使用常量缓冲区完成操作之前，CPU 不应尝试更新任何常量缓冲区。如果只有一个常量缓冲区，CPU 必须一直处于阻止状态，直到 GPU 完成绘图，这是不现实的，因为常量需要在几次绘图调用中更新。因此，本示例使用多个常量缓冲区，以便在 GPU 绘图时，CPU 可以继续将常量发送到 GPU。

示例非常简单，每个帧有一个固定数目的绘图调用，这在编译时是已知的。本示例为每个绘图调用创建一个缓冲区，再乘以交换链中的后台缓冲区数量。这个数字保证了始终有一个空闲的缓冲区供 CPU 写入。所有常量缓冲区都存储在 Sample::CreateDeviceDependentResources 中创建的单个相邻的上载缓冲区中。将立即映射上载缓冲区，以获取 CPU 地址空间和 GPU 地址空间的基本内存地址。

在 Sample::Render 方法中，将常量写入常量上载堆的 CPU 基址之外的索引位置。相同的索引与 GPU 地址结合在一起，可通过调用 ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView 将缓冲区绑定到管道。索引必须考虑到后台缓冲区的节奏和绘制计数。有关详细信息，请参阅 Sample::Render 实现。

## CPU/GPU 同步

如果 CPU 能够以比 GPU 处理命令更快的速度向 GPU 发出命令列表，那么最终 CPU 将不得不等待 GPU 赶上来。本示例以循环方式使用常量缓冲区内存，这意味着在固定时间段后将重新使用缓冲区槽。一般情况下，在重用共享资源之前，请务必使用某种同步策略来确保资源未使用。 本示例使用 ID3D12Fence 来同步 CPU 和 GPU。CPU 将命令插入命令队列，用帧索引值向防护对象“发送信号”。在 GPU 上处理信号命令后，CPU 将可以立即看到提供的帧索引值。这使 CPU 可将当前帧索引与 GPU 发出信号的最后一个帧索引进行比较，以确定 GPU 与 CPU 相比落后多少。如果 GPU 帧计数和 CPU 帧计数之间的差值超出后台缓冲区的数量，则 CPU 将需要等待。

# 隐私声明

在编译和运行示例时，示例可执行文件的文件名将发送给 Microsoft，用于帮助跟踪示例使用情况。要选择退出此数据收集，你可以删除 Main.cpp 中标记为“示例使用遥测”的代码块。

有关 Microsoft 的一般隐私策略的详细信息，请参阅《[Microsoft 隐私声明](https://privacy.microsoft.com/en-us/privacystatement/)》。