

简单光照示例 (DX12)

*此样本与 2016 年 8 月的 Xbox One XDK 兼容。*

# 描述

本样本演示如何创建静态 Direct3D 12 顶点、索引和常量缓冲区，以绘制使用静态和动态Lambertian 光照点亮的索引几何体。

该示例渲染了一个由两个灯光点亮的大型立方体，一个白色和一个红色，也表示为立方体。白灯静止，而红灯围绕中央立方体运行。中央立方体也旋转。该动作允许您从不同角度观察彩色灯光的效果。



# 使用样本

|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 退出 | 查看按键 |

# 实施说明

## 着色器

## 该示例使用三个着色器渲染场景 - 顶点着色器（“TriangleVS”）和两个像素着色器（“LambertPS”，“SolidColorPS”。）将已编译的着色器块载入 CreateDeviceDependentResources，然后在为每个着色器组合创建管道状态对象时引用。 所有着色器都在同一个HLSL包含文件中定义，“SimpleLighting.hlsli”和三个存根着色器包含此文件。为不同的入口点编译每个存根着色器，以便创建三个着色器blob。

## 管道状态对象(PSO)

简单光照示例有两个唯一着色器组合：第一个是 TriangleVS 和 LambertPS，第二个是 TriangleVS 和 SolidColorPS。在 DirectX 12 中，您需要为每个唯一着色器组合创建管道状态对象 (PSO)。顾名思义，PSO 封装了使用一组特定着色器可能进行的多次绘制调用所需的所有管道状态。PSO 包括了根签名、管道各阶段的着色器、状态设置，如光栅器状态、深度模板状态、混合状态等（有关详细信息，请参阅 MSDN 上的文档。）

## 根签名

根签名定义了与图形管道绑定的资源类型以及资源展示方式。根签名类似于 API 函数签名，其中描述了参数类型、参数顺序和排列，但未定义任何实际参数实例。根参数是与根签名元素相对应的实际数据的实例。对于本示例的顶点着色器，着色器常量只需要单个结构，因此根签名非常简单。根签名包含 ConstantBufferView 类型的单个根参数。

## 几何

## 此场景的几何结构由静态顶点和索引数组组成，其中包含表示立方体六个四边形的 24 个顶点的数据。这两个数组在 Sample::CreateDeviceDependentResources 中声明，其中 ID3D12Device::CreateCommittedResource 立即将其用于为缓冲区创建 ID3D12Resources。为简单起见，本示例使用 D3D12\_HEAP\_TYPE\_UPLOAD，因为这可让您创建每个资源时只需单个步骤便可使用数据对其进行初始化。但是，\_UPLOAD 堆是几何数据的次优位置。针对几何数据使用 D3D12\_HEAP\_TYPE\_DEFAULT 是更高效的实现。如要初始化 \_DEFAULT 堆，您需要使用 \_UPLOAD 堆，这样最终会使用两个堆，使得实现变得复杂。

创建几何结构缓冲区后，本示例可为顶点创建 D3D12\_VERTEX\_BUFFER\_VIEW，为索引创建 D3D12\_INDEX\_BUFFER\_VIEW。通过调用 ID3D12GraphicsCommandList::IASetVertextBuffer 和 ID3D12GraphicsCommandList::IASetVertextBuffer 设置输入装配时，Sample::Render 中会使用这些视图。

## 管理着色器常量

对于这个非常简单的场景，所有着色器常量都集中在一个包含以下内容的常量缓冲区中：

* 世界，视图和投影矩阵
* 光线方向和颜色
* 纯色

对于更复杂的场景，通常会根据常量更新的频率将常量拆分为多个缓冲区。

大立方体和红灯均为动态，因此着色器常量需要在绘制调用之间每帧更新多次。回想一下，根签名包含一个 ConstantBufferView 类型的根参数，该参数需要引用着色器常量的副本，以便用于每次绘制调用。CPU 和 GPU 并行运行，因此 CPU 不应尝试更新任何缓冲区直至 GPU 结束使用。如果只有一个常量缓冲区，则在 GPU 完成绘制前必须阻塞 CPU，这不现实，因为常量需要针对多个绘制调用更新。因此，本示例使用了多个常量缓冲区，以便 CPU 可在 GPU 绘制时继续向 GPU 发送常量。

本示例很简单，每帧的绘制调用数量固定，该数量在编译时为已知。本示例为每个绘制调用创建一个缓冲区，乘以交换链中的后台缓冲区数。这个数量可确保始终存在空闲缓冲区可供 CPU 写入。所有常量缓冲区均储存在 Sample::CreateDeviceDependentResources 中创建的单个连续上传缓冲区中。上传缓冲区立即被映射以获取 CPU 地址空间和 GPU 地址空间的基本内存地址。

在 Sample::Render 方法中，常量被写入到根据常量上传堆 CPU 基址建立索引的位置。相同的索引加上 GPU 地址，通过调用 ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView 将缓冲区绑定到管道。该索引必须考虑后台缓冲区节奏以及绘制计数。如欲了解更多详细信息，请参阅 Sample::Render 的实现。

## CPU/GPU 同步

如果 CPU 向 GPU 发送命令列表的速度超过了 GPU 处理命令的速度，那么最终 CPU 需要等待 GPU 完成。本示例以轮循方式使用常量缓冲内存，这意味着缓冲槽将在固定时段后重用。通常，在重用共享资源之前，需要使用某种同步策略确保该资源未在使用中。 本示例使用 ID3D12Fence 以同步 CPU 和 GPU。CPU 将一个命令插入命令队列，以使用帧索引值“发信号通知”fence 对象。GPU 上处理了信号命令后，所提供的帧索引值将立即对 CPU 可见。这样，CPU 能够将当前帧索引与由 GPU 发信号通知的最后帧索引进行比较，以确定 GPU 相对于 CPU 的落后差距。如果 GPU 帧计数和 CPU 帧计数之间的差异超过了后台缓冲区数量，则 CPU 将需要等待。

# 更新历史记录

2016 年 5 月，已弃用 ATG 示例框架的所有依赖项删除。

2016 年 6 月，DirectX 12 版本创建。2017 年 4 月，它更新为在 Xbox One X 上运行时支持 4k 交换链。

# 隐私声明

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

有关 Microsoft 隐私政策的更多信息，请参阅 [Microsoft 隐私声明](https://privacy.microsoft.com/zh-cn/privacystatement/)。