

Simple Lighting Sample

# *This sample is compatible with the Windows 10 October 2018 Update SDK (17763)*

# Description

This sample demonstrates how to create a static Direct3D 11 vertex, index, and constant buffer to draw indexed geometry lit by using static and dynamic Lambertian lighting.

The sample renders a large cube that is lit by two lights, one white and one red, also represented as cubes. The white light is stationary while the red light rotates around the central cube. The central cube also rotates. The motion allows you to observe the effects of the colored lights from different angles.



# Using the sample

|  |  |  |
| --- | --- | --- |
| Action | Gamepad | Keyboard |
| Exit | View Button | Esc |

# Implementation notes

The sample uses three shaders to render the scene – a vertex shader (“TriangleVS”) and two pixel shaders (“LambertPS”, “SolidColorPS”.) The compiled shader blobs are loaded in CreateDeviceDependentResources and then referenced when creating the shader resources. All shaders are defined in the same HLSL include file, “SimpleLighting.hlsli” and three stub shaders include this file. Each stub shader is compiled, for a different entry point, in order to create the three shader blobs.

The geometry for the scene is composed of static vertex and index buffers which are each filled with data for 24 vertices representing six quads of a cube. These two buffers are created in CreateDeviceDependentResources and immediately filled with the data provided in the initialization data of the D3D11\_SUBRESOURCE\_DATA structure. This is the most efficient method of initialization for DirectX 11.

For this very simple scene, all the shader constants are lumped together into a single constant buffer which contains the following:

* World, View and Projection Matrices
* Light directions and colors
* Solid color

For a more complex scene, you would normally split the constants into multiple buffers depending on how frequently the constants are updated.

Since the large cube and the red light are animated, some of the constants must be updated every frame. The constant buffer is updated using the following strategy:

1. Call ID3D11DeviceContext::Map, using D3D11\_MAP\_DISCARD, to create a D3D11\_MAPPED\_SUBRESOURCE.
2. Copy the values for all the constants into the mapped resource
3. Unmap the resource before calling DrawIndexed

When using D3D11\_MAP\_DISCARD

* The driver essentially allocates a new buffer and so you should assume that you must update all the constants in the buffer that the shader will use. Compare with D3D11\_MAP\_WRITE\_NO\_OVERWRITE where the buffer could contain valid data from a previous update.
* If the previously mapped buffer is still in use by the GPU, you will get a new buffer. If you over use D3D11\_MAP\_DISCARD, you could end up stalling since you will have to wait for more memory to become available.

# Update history

May 2016, removed all dependencies on the deprecated ATG Sample Framework.

# Privacy statement

When compiling and running a sample, the file name of the sample executable will be sent to Microsoft to help track sample usage. To opt-out of this data collection, you can remove the block of code in Main.cpp labeled “Sample Usage Telemetry”.

For more information about Microsoft’s privacy policies in general, see the [Microsoft Privacy Statement](https://privacy.microsoft.com/en-us/privacystatement/).