

SimplePBR Sample

# *This sample is compatible with the Windows 10 Creators Update SDK (15063) / August 2016 Xbox One XDK.*

# Description

This sample demonstrates physically based rendering (PBR) on UWP and Xbox One XDK using DirectX 12. The sample implements forward-rendered Disney-style PBR as a standalone shader with the following parameters:

1. Albedo: base RGB color with no lighting
2. Normal map: uncompressed 3 channel normal map (Y positive),

And an RMA map which specifies:

1. Roughness: [0..1], Normal distribution implying specular highlight size and shape. Roughness is scaled according to Disney paper.
2. Metallic: (Usually 0 or 1, may blend), Controls index of refraction, specular vs diffuse distribution of albedo.
3. Ambient occlusion: A value [0..1] that scales specular and diffuse contribution.

The parameters may be expressed as constants only or as textures only (but not a mix). The shader supports image based lighting (with precomputed diffuse and specular maps) and directional lights.

For more details of PBR, see the implementation/references section at the end of this document.



# Building the Sample

Build and run the SimplePBRUWP12 or SimplePBRXbox12 solutions. All the assets are shared in the “Assets” subdirectory, and the shared code (including shader source) is in the “Shared” subdirectory. The shaders are compiled by the Visual Studio 2015 project, no external build steps are required.

# Using the sample

On UWP, use the mouse (mouse wheel, right click) and keyboard (W, A, S, D keys) to move the camera around the scene.

Gamepad camera control is also available on both UWP and Xbox One: use the left and right stick to change camera orientation and radius.

# Implementation notes

The PBREffect class wraps the implementation of the shader. The shader has two configurations – constant and textured. The constant configuration is mainly for debugging. In the textured configuration, the input parameters (Albedo, and Roughness, Metallic, AO) are specified as textures.

To create a textured shader, use the EffectFlags enum:

m\_effect = std::make\_unique<ATG::PBREffect>(device, EffectFlags::Texture, pipelineState);

To set texture parameters, simply pass in descriptors for each texture and a sampler:

m\_effect->SetSurfaceTextures(m\_descriptors->GetGpuHandle(AlbedoIndex),

m\_descriptors->GetGpuHandle(NormalIndex),

m\_descriptors->GetGpuHandle(RoughnessMetallicAOIndex),

commonStates->AnisotropicWrap());

The shader compiles as part of the Visual Studio project, and is broken up into three files

1. PBREffect\_VSConstant – shared vertex shader
2. PBREffect\_PSConstant – constant parameter pixel shader
3. PBREffect\_PSTextured – textured parameter pixel shader

There are also two HLSL includes

1. PBREffect\_Math – shared math functions for BRDF etc.
2. PBREffect\_Common – Root signature, constants and shared lighting function “PBR\_LightSurface”.

## Lighting

PBREffect supports both directional and image based lighting. The caller must supply a precomputed irradiance texture (for diffuse environment lighting) and a radiance texture (for specular environment lighting). The textures should be DDS cubemaps in a HDR format.

The caller must also specify the number of MIP levels in the radiance texture. See “[AMD Cubemapgen for physically based rendering](https://seblagarde.wordpress.com/2012/06/10/amd-cubemapgen-for-physically-based-rendering/)” for more details on generating precomputed maps for PBR.

m\_effect->SetIBLTextures(

m\_descriptors->GetGpuHandle(m\_radTexDescIndex),

m\_radianceTexture->*GetDesc*().*MipLevels*,

m\_descriptors->GetGpuHandle(m\_irrTexDescIndex),

m\_commonStates->AnisotropicClamp());

Optional, the caller can also specify directional lights using the SetLight\* methods. The shader will blend the directional and image lighting.

## References

<https://www.allegorithmic.com/system/files/software/download/build/PBR_Guide_Vol.1.pdf>

<https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf>

<http://blog.selfshadow.com/publications/s2015-shading-course/>

<http://graphicrants.blogspot.com/2013/08/specular-brdf-reference.html>

<https://github.com/dariomanesku/cmftStudio>

## UWP on Xbox

To support DirectX 12 on Xbox One, a UWP app must have its app type set to ‘Game’. Otherwise, only the software device (WARP12) is available on developer consoles, which is unsupported for retail consoles. During development the app type can be set via DevHome. This sample uses the [expandedResources](https://msdn.microsoft.com/en-us/library/windows/desktop/mt808808.aspx) restricted capability to achieve this by default, but can be removed after setting the package app type to ‘Game’. Note that apps submitted to the Windows Store will fail validation if using this restricted capability.

# Update history

October 2016 initial release. Updated in June 2017 to utilize the [DirectX Tool Kit tone-mapper](https://github.com/Microsoft/DirectXTK12/wiki/PostProcess).

Updated in October 2017 to support HDR10 output on Xbox One S / Xbox One X as well as a Windows 10 Creators Update PC with HDR output configured.

# 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/).