Physically Based Shading in Unity

Aras Pranckevičius
Rendering Dude

This is about physically based shading in upcoming Unity 5, and all things that fall out of that.

I am Aras and have been working on Unity graphics since 2006. @aras_p on the twitterverse.

Outline

- New built-in shaders in Unity 5
  - What, how and why
- And all related things

Note: everything in this talk describes what is planned for Unity 5, which is in very early alpha stage right now. We really want to ship all this, but if things go wrong, some features might get pulled from 5.0 release.

Shaders in Unity 4.x

- A lot of good things are *available*
  - Marmoset Skyshop
  - Shader Forge
  - Alloy physical shaders
- A lot of good things are *possible*
  - If you know how to write shaders, that is

Marmoset Skyshop: image based lighting system and shaders. https://www.assetstore.unity3d.com/#/content/8880

Shader Forge: awesome graphical shader editor with options for physically based shading. https://www.assetstore.unity3d.com/#/content/14147

Alloy: physical shader framework. https://www.assetstore.unity3d.com/#/content/11978

Want this

Done with one built-in shader

Unity 4.x has a bunch of built-in shaders (Diffuse, Bumped Specular, …). Many of them are just variations of essentially the same “blinn-phong shader”.

We want to both improve the shading model (oldskool Blinn-Phong made sense in 2005… not so much in 2014), and improve workflow.

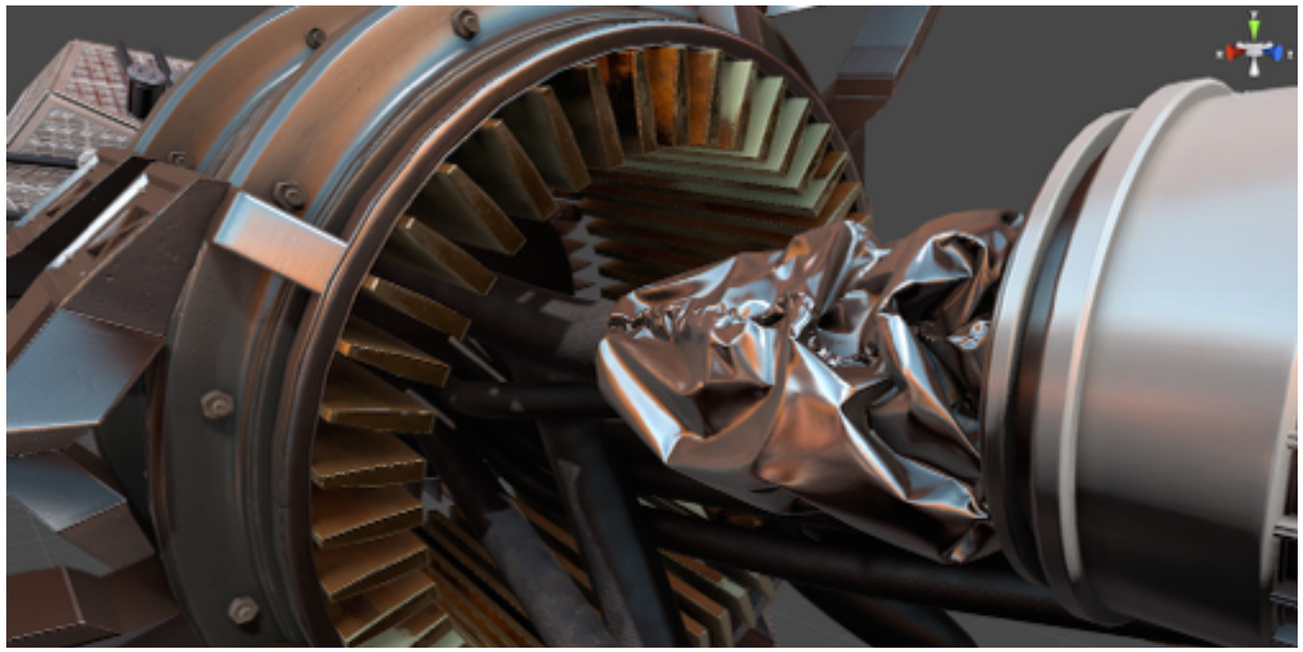
We want the scene like this to be made with built-in shaders & features.

This scene has all textures from Quixel’s Megascans HDR scanning (http://dev.quixel.se/megascans) by the way.

Shaders in Unity 5

- New built-in shaders
  - Physically based shading
  - Improved UI & workflow
- Related things
  - Deferred shading, Reflection probes, HDR, …
- Built-in could cover 80% of use cases
  - Still possible to write your own, like always

Physically Based Shading



Physically Based Shading (PBS)

- Try to model how light actually behaves
  - Instead of ad-hoc models that may or might not work
- Movies have moved to it
- Games are moving to it
- Does not have to mean“photo-realism”!
  - See Pixar

Tons of background information on PBS in recent siggraph courses, for example http://blog.selfshadow.com/publications/s2013-shading-course/ and http://blog.selfshadow.com/publications/s2012-shading-course/

At Unity, our first big experience with PBS was in Butterfly Effect R&D demo (http://unity3d.com/pages/butterfly) in 2012.

PBS motivation

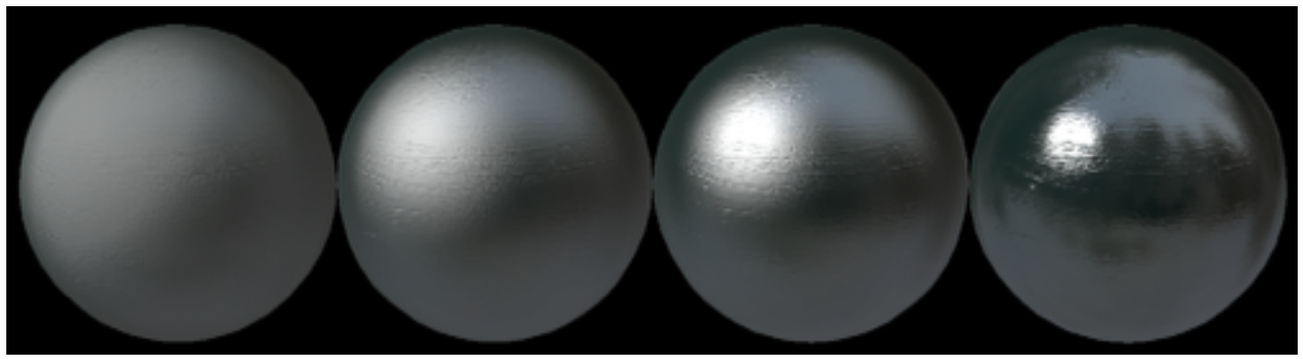
- Consistent, plausible look under different lighting conditions
- *Less* material parameters
- All parameter values behave “sensibly”
- More interchangeable data
  - From 3D / material scanning etc.

Lighting Conditions

Want same content to look good

Energy conservation

- Don’t reflect more light than you receive
- Sharper reflections = stronger; more blurry reflections = dimmer
- Specular takes away from diffuse



Specular everywhere

- Everything has specular
  - Really! Even if a tiny amount

“Everything is Shiny” http://filmicgames.com/archives/547 (John Hable 2010)

Fresnel all the things

- Everything has Fresnel
  - Fresnel: surface becomes more reflective at grazing angles
  - *All* surfaces approach 100% reflective!
    - Kind of; Fresnel is more complicated on rough surfaces

“Everything has Fresnel” http://filmicgames.com/archives/557 (John Hable 2010)

Image Based Lighting (IBL)

- *Think Marmoset Skyshop*
- Capture environment into a cubemap
- Blur mip levels in a special way for varying roughness
- Voilà! Good looking reflections
- Place a bunch of them in the level, objects pick up best matching ones

Blur mip levels: every mip step increases the angle of specular. We use a non-linear angle curve to have more detail in low-angle region.

New Standard Shader

New Standard Shader

- Good for most everyday materials
  - Metals, Plastics, Wood, Concrete, Cloth, …

Primary Inputs

- Diffuse color
- Specular color
- Surface smoothness
- Normal map

Input: Diffuse

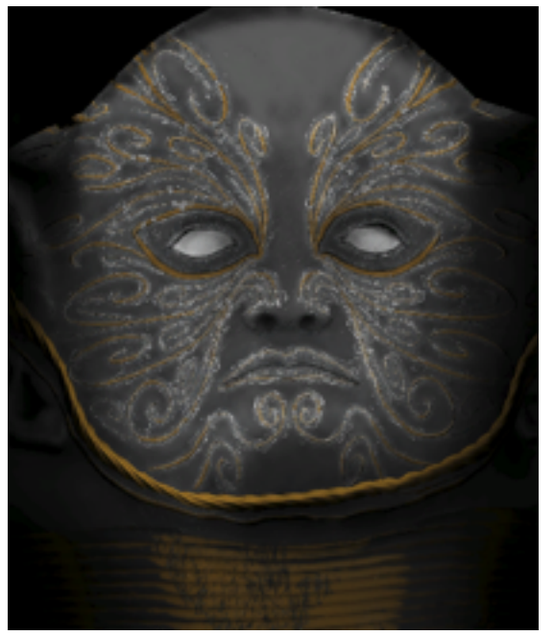
- a.k.a. Albedo
- Should not have lighting!
- Metals have no (black) diffuse
  - Rust etc. can make it not pure black

Input: Normals

- The usual tangent space normal map

Input: Specular

- Only metals have colored specular
- Non-metals: gray and quite dark
- Very few details in a single material



Input: Smoothness

- a.k.a. glossiness
- or opposite of “roughness”
- 0=rough, 1=smooth
- Interesting detail in this map

Optional inputs

- Emission
- Ambient Occlusion
- Detail albedo/normal maps
- Heightmap (for parallax)

Automatic inputs

- Specular reflection cubemap
  - From reflection probes or global in scene
- Light probes, lightmaps, …

Shading maths

- Microfaced based BRDF
- Inspired by Disney's 2012 research
  - Single smoothness for diffuse & specular
  - Specular takes away from diffuse
- Cook-Torrance, Schlick Fresnel, Blinn-Phong NDF

If you don't know any of these words, do not worry. They describe math done by the shader to make pretty pixels.

Disney's stuff: see "Physically-Based Shading at Disney" (Brent Burley 2012) here http://blog.selfshadow.com/publications/s2012-shading-course/

Light attenuation

Here we’re going over shading step by step

Ambient Occlusion

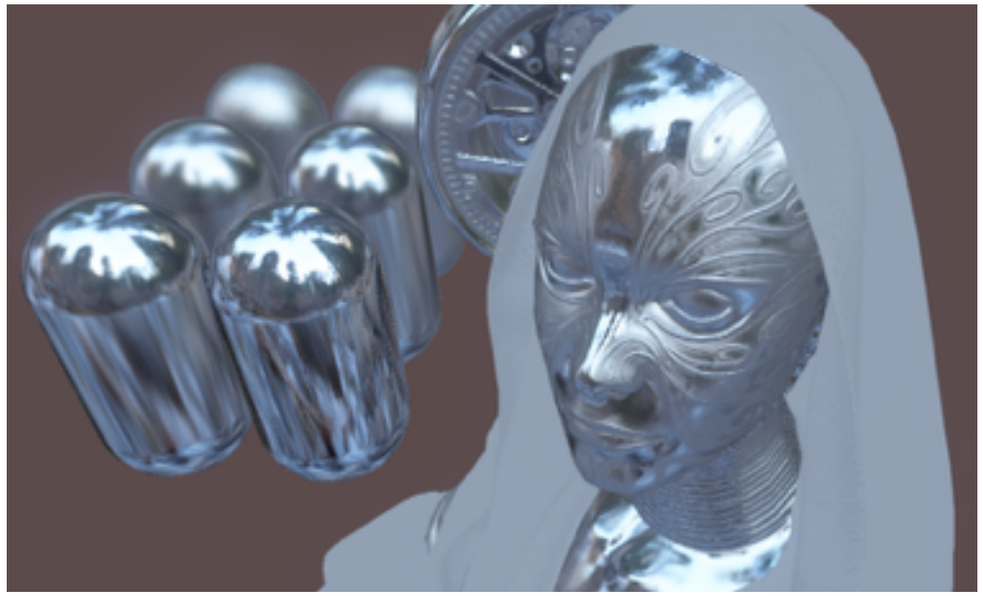
AO from baked texture

In theory could also plug screen-space AO or other “global AO” here.

Environment

Just raw cubemap

Environment coming from reflection probe.



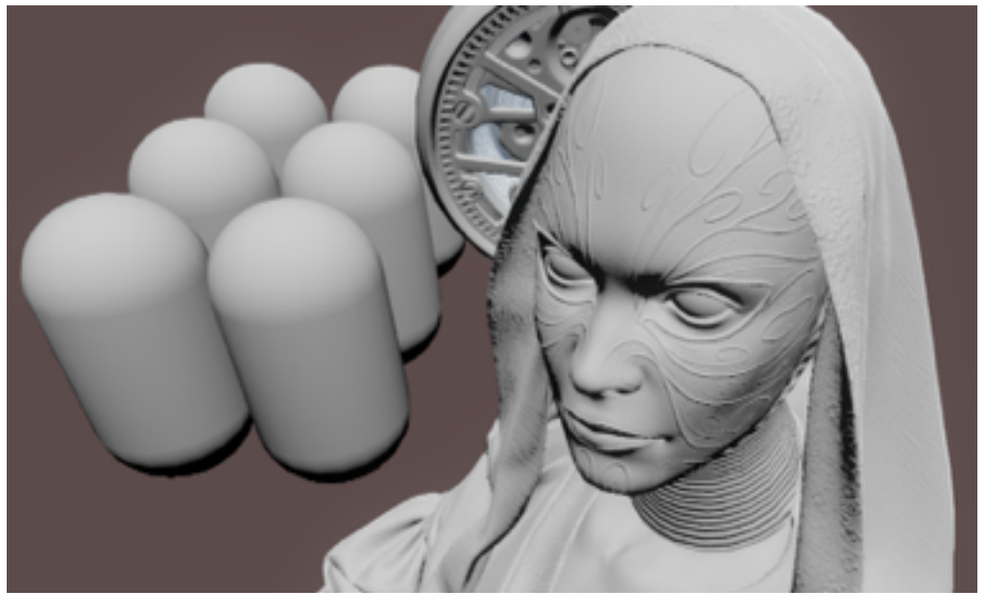
Environment

With proper blur based on smoothness

Cubemap in the reflection probe has mip levels blurred in special way (“specular convolution”) so that surfaces of different smoothness can pick up appropriately blurred reflection.

Environment

And with ambient occlusion



Diffuse factor

Specular factor

Diffuse part

Specular from light

Reflection for metals

Coming from the environment

Fresnel reflection for non-metals

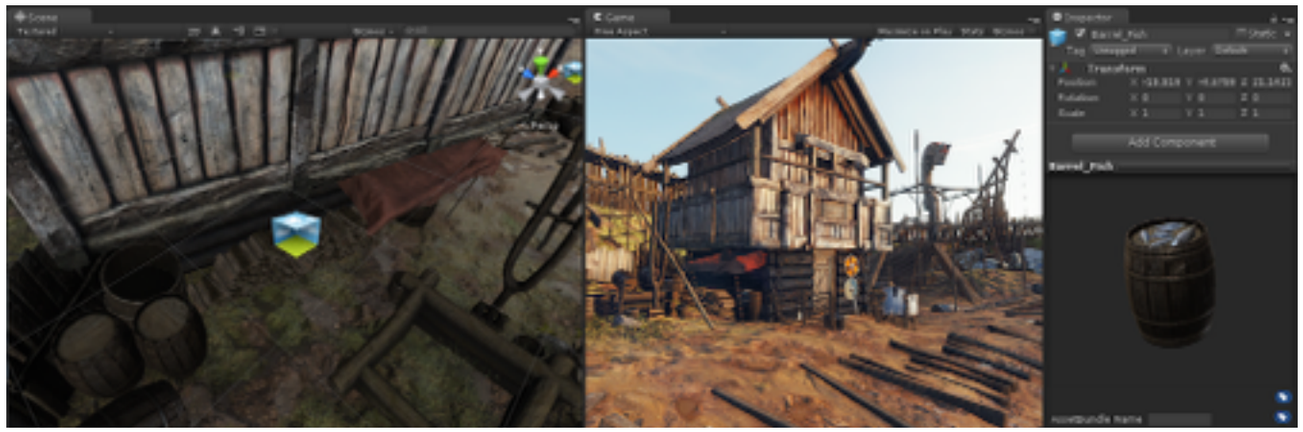
Coming from the environment

Diffuse+Specular

Everything

Diffuse + Specular + that Fresnel I could not explain in simple words

Workflow



One shader

- Less guessing at which shader to use
  - “So what is alpha bumped unlit reflective again?”
- No “holes” in variants
  - “I need emission, parallax & alpha, but there is no such shader”

Configurable

- Enable features by assigning textures
- Compact UI
- Additional controls next to relevant textures

We’re trying a different UI than it used to be in Unity 4.x. Primary motivation:

- More compact
- Controls relevant to textures (e.g. normal map strength etc.) are next to them
- Shift|Ctrl-clicking on the texture slot brings up texture preview popup

Configurable

- Internally, lots of shader variants
  - No emission -> uses faster variant
  - No detail texture -> uses faster variant
- Unused variants not included into game data
  - Otherwise shader would take 200MB…

Related Things

Deferred Shading

- Old deferred lighting (a.k.a. light pre-pass) had minimal g-buffer, and two geometry passes
  - Not enough space in g-buffer for new shader
  - Two geometry passes not nice either
- So we’re making full deferred shading
  - Old one stays as “legacy”

Unity 3/4 deferred lighting (a.k.a. “light pre-pass”) has really tiny g-buffer (just normals & glossiness). The advantage of it is that it does not require multiple render targets. Disadvantage is two geometry passes, and hard to express any more sophisticated shading model.

So we’re making “full deferred shading” option, with one geometry pass and multiple render targets to store the g-buffer information.

Deferred Shading

- G-buffer layout
  - RT0: diffuse color (rgb)
  - RT1: specular color (rgb), smoothness (a)
  - RT2: world normal (rgb; 10.2)
  - RT3: emission/light buffer; FP16 when HDR
  - Z-buffer: depth
- 160bpp (ldr), 192bpp (hdr)
  - Quite “fat”
  - Likely not final yet

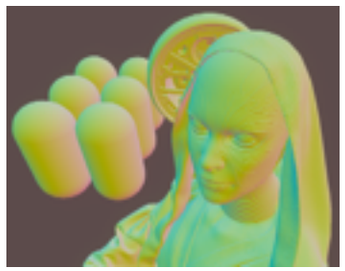
This is the current WIP g-buffer layout. Uses 4 MRTs (all 32 bit, except emission/light buffer which can be 64 bit when using HDR).

Since it needs MRTs, that makes it require OpenGL ES 3.0 on mobile, and not-too-old GPU (i.e. not a 10 year old one) on PC.

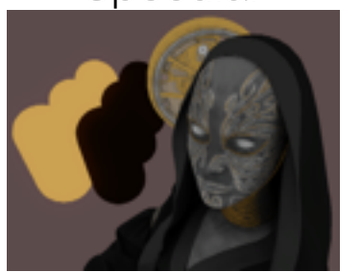
Deferred Shading

Diffuse

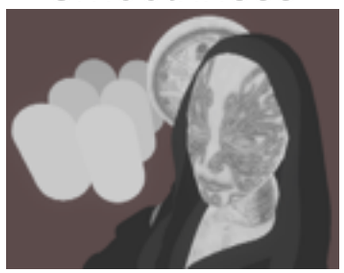
Normals



Specular

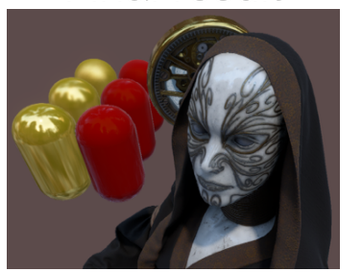


Smoothness



Depth

Final result



Deferred Shading

- Thinking about extensibility
  - Custom g-buffer layouts
  - Custom stuff rendered around (g-buffer, lighting) passes
- Very much WIP area right now

Reflection Probes

- Similar to LightProbes, just for reflection
- A box with size placed in the level
  - Baked cubemap with specular mip levels
  - Can also be realtime rendered (expensive)
- Objects pick up best probe for them
  - And data is fed into shaders

Reflection Probe

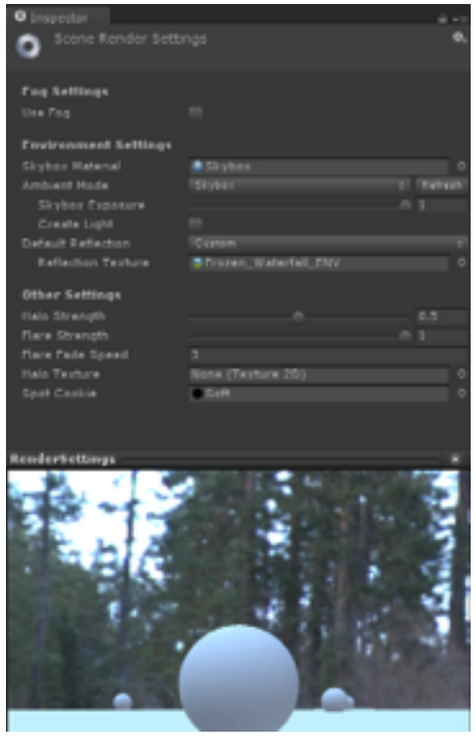
Providing environment for nearby objects

Reflection Probe

Final shaded result

Cubemaps

- Improved cubemap generation from textures
- Cubemap texture compression
- Specular & diffuse convolution for IBL
- Large cubemaps? 64 bit editor ;)
- RenderSettings get a default reflection cube
  - Also directional ambient, better inspector, …



HDR

- Scene view can be HDR now
  - When main camera is HDR
  - Uses same tonemapper as main camera
- HDR texture import
  - Encode to RGBM, including cubemaps
  - .hdr/.exr formats

Performance of new shader?

- Right now ok for decent PC & consoles
- Working on mobile/low-end fallbacks
  - Minimalist Cook-Torrance
  - Math baked into lookup texture; smoothness in mip chain
  - Fresnel approximation with 4.0 power
  - tex*Lod -> tex*Bias for DX9 SM2 / GLES2.0
  - No detail texture
  - …

Minimalist Cook-Torrance: http://www.thetenthplanet.de/archives/255 (Christian Schüler 2011)

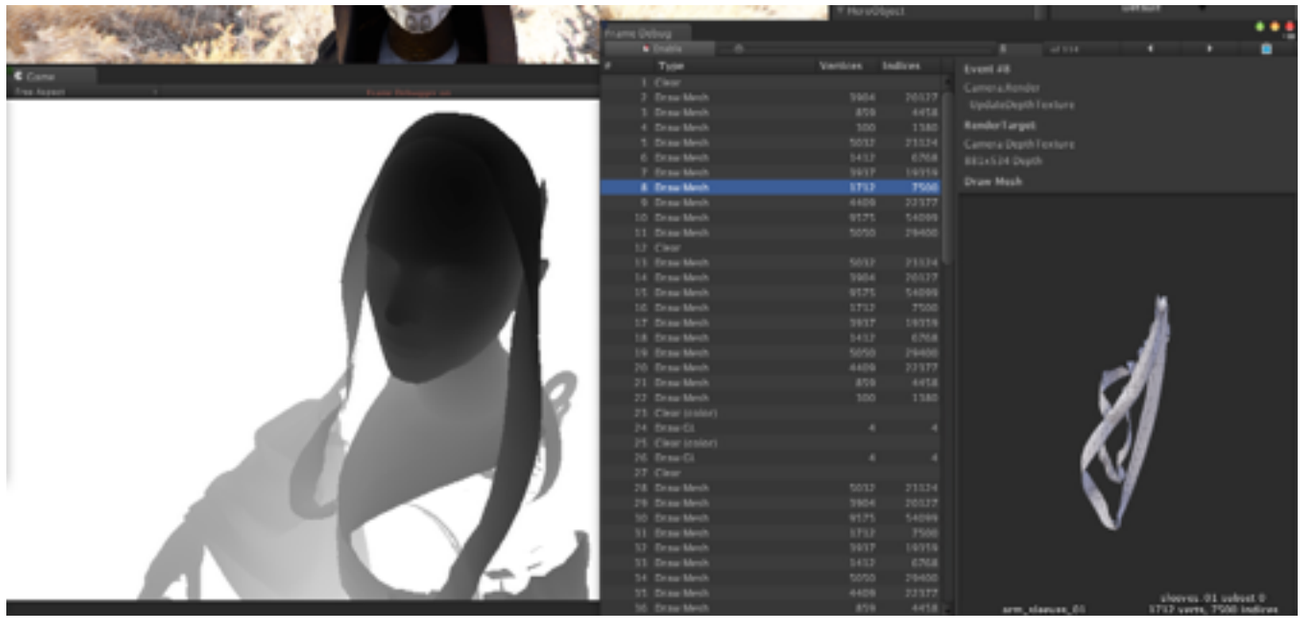
PBS math from lookup textures: http://framebunker.com/blog/static-sky-unite-presentation/ (Nicholas Francis 2013)

Preemptive answers

- Integration with new GI stuff?
  - Yes.
- Area lights?
  - Probably not in 5.0, but yes, will need them.
- What if I don’t need all this?
  - You can write your own shaders like you always could.
- More shaders?
  - Not in 5.0, but having “clear coat” (car paint etc.) & skin built-in would make sense.

Questions?

Random Bonus Feature



While we’re on topic of graphics & rendering, here’s a random other Unity 5.0 feature: Frame Debugger.

Frame Debugger

- Step through draw calls and see exactly how the frame is rendered
- Current render target, mesh being rendered etc.

You can already do something like this in external tools like GPA, VS2012, NSight, PIX, RenderDoc etc., but having a slimmed down version integrated into Unity for much less hassle would be useful, we thought.

This is particularly useful to see what is rendered into render textures (reflections, depth textures, shadow maps etc.) or which things get batched together.