

GPU PROGRAMMING FOR VIDEO GAMES

Physically-Based Rendering



Prof. Aaron Lanterman

School of Electrical and Computer Engineering

Georgia Institute of Technology



PBR does not imply “photorealistic”



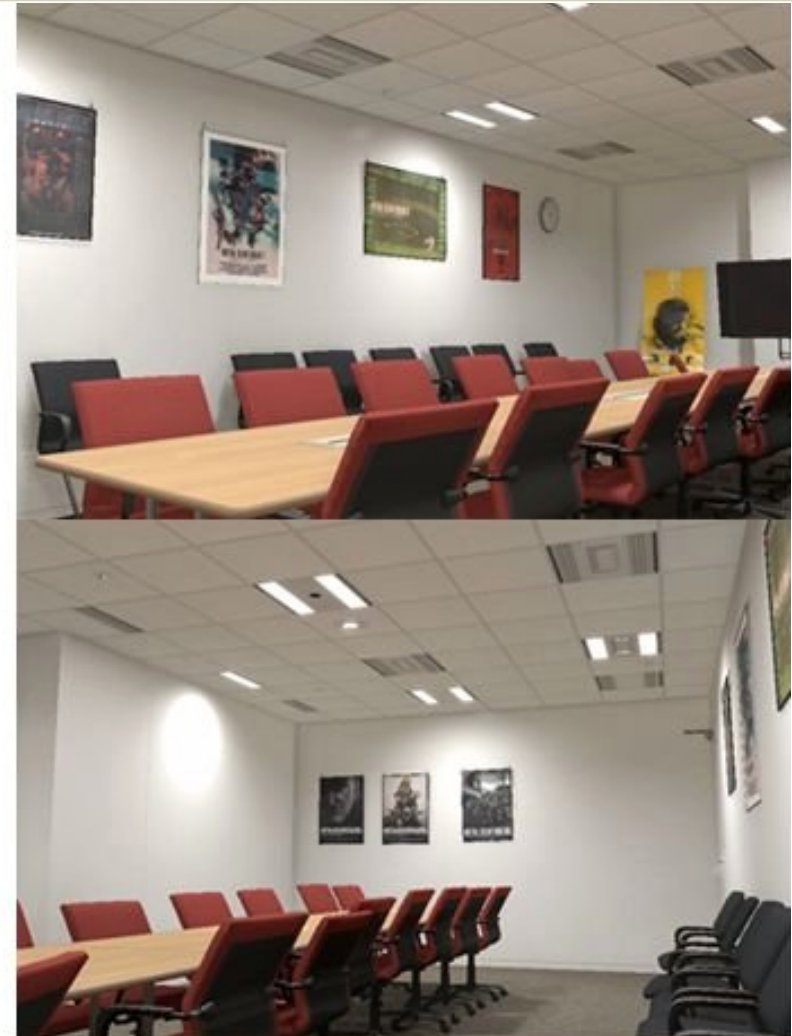
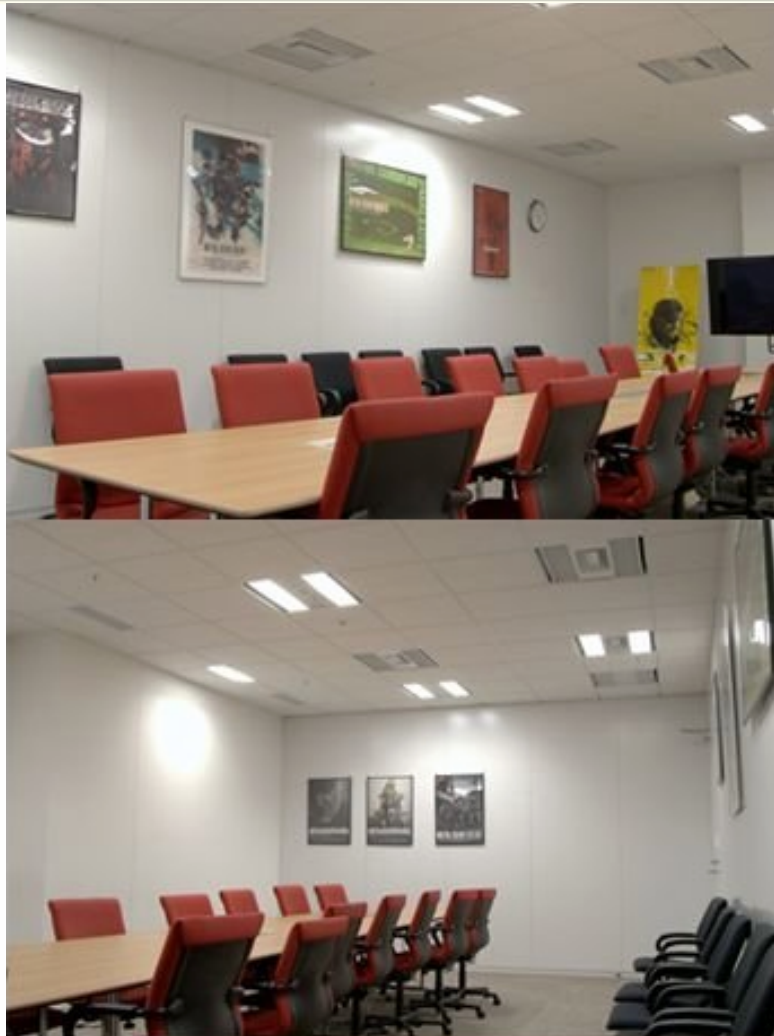
B. Burley,
“Physically Based
Shading at Disney”



“Physically Based
Lighting at Pixar,” C.
Hery and R. Villemin

From <http://blog.selfshadow.com/publications/s2012-shading-course/>
and <http://blog.selfshadow.com/publications/s2013-shading-course/>

But photorealism implies PBR!



Kojima Productions Fox Engine challenge

From http://andriasang.com/con063/fox_engine_lighting

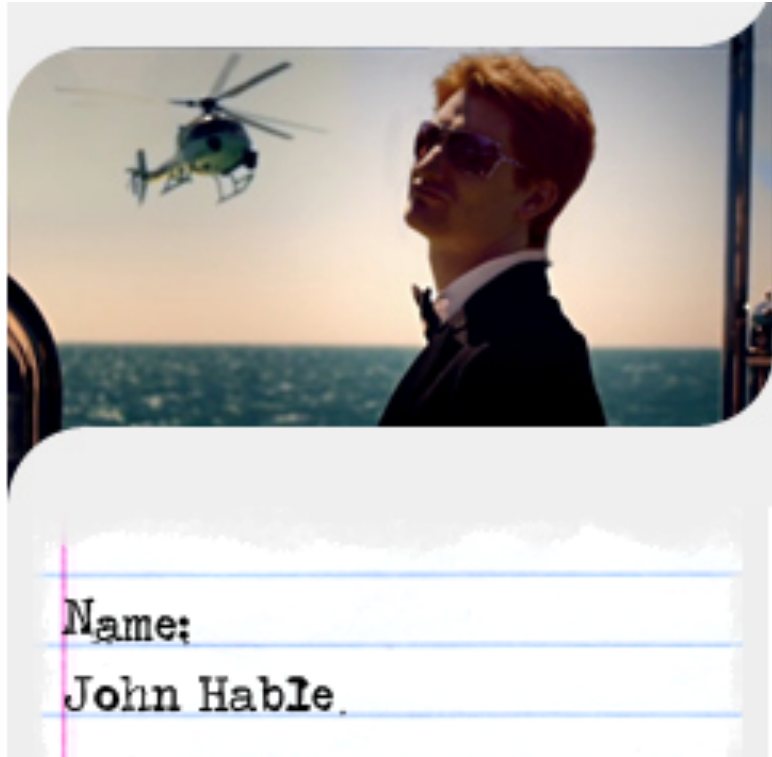
Which is real, which is rendered?



Kojima Productions Fox Engine challenge

From http://andriasang.com/con063/fox_engine_lighting

Credit where it is due



filmicgames.com
filmicworlds.com

BS&MS in CS from GT

Bio:

Little is known. Reliable sources have confirmed sightings at Georgia Tech, Electronic Arts in Vancouver and Los Angeles, and Naughty Dog. He is rumored to have worked on Uncharted 2 and Steven Spielberg's project, LMNO. Several eyewitnesses claim to have seen him compressing Tiger Woods's face, but said reports are dismissed as speculation.

Elements of PBR

- Linear space lighting
- Energy conservation
- Reciprocity
- Metallic/dielectric distinction
- “Everything is shiny” (i.e., has specular)
- “Everything has Fresnel”
- Bonus: high definition render buffers and tonemapping to handle high dynamic range



Elements of PBR

- Linear space lighting
- Energy conservation
- Reciprocity
- Metallic/dielectric distinction
- “Everything is shiny” (i.e., has specular)
- “Everything has Fresnel”
- Bonus: high definition render buffers and tonemapping to handle high dynamic range

Important for
Global Illumination



Hable on linear space lighting



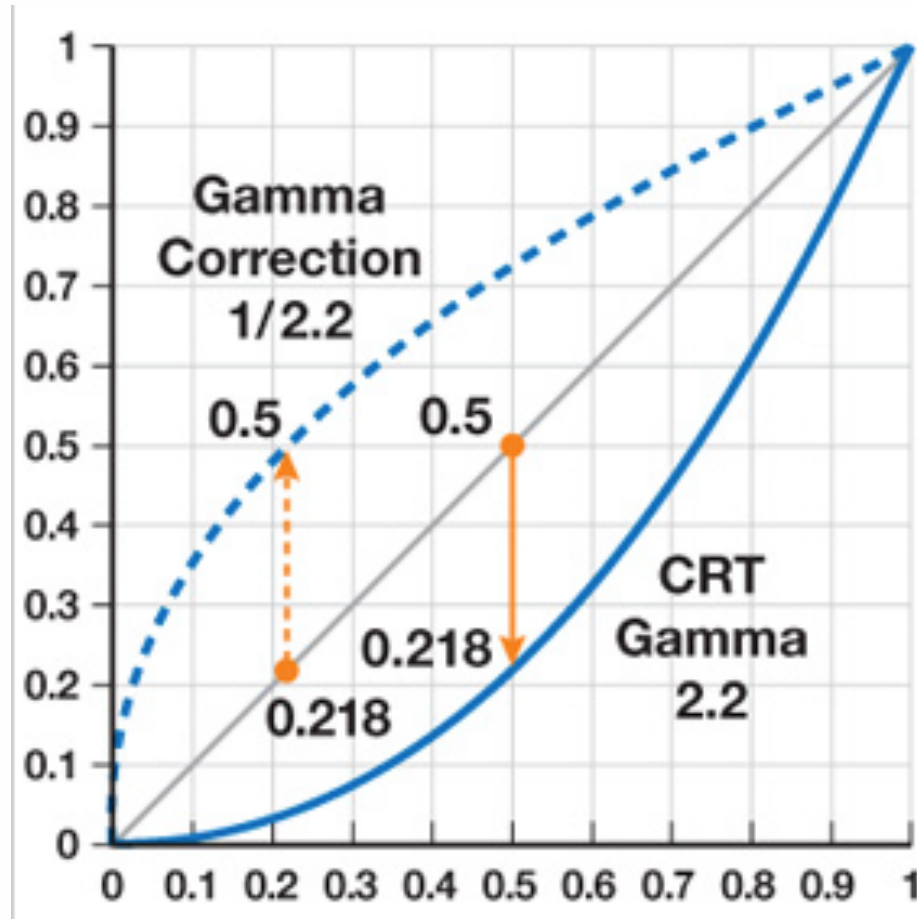
“Linear-space lighting is the single most important thing for graphics programmers and technical artists to know.

It’s not that hard, but for whatever reason, no one really teaches it.

Personally, I got a BS and MS at Georgia Tech, took basically every graphics class they had, and didn’t hear about it until I learned about it from George Borshukov.”

– John Hable

Gamma curves



From http://http.developer.nvidia.com/GPUGems3/gpugems3_ch24.html

DIY linear space (helpers)

```
inline float3 DeGamma (in float3 c) {  
    // return c*c;  
    return pow(c, 2.2);  
}
```

```
inline float3 ReGamma (in float3 c) {  
    // return sqrt(c);  
    return pow(c, 1/2.2);  
}
```


DIY linear space in shader

```
#pragma surface surf BlinnPhong finalcolor:finalgamma
```

• • •

```
void finalgamma (Input IN,  
    SurfaceOutput o, inout fixed4 c) {  
    c = fixed4(ReGamma(c.xyz),c.a);  
}
```

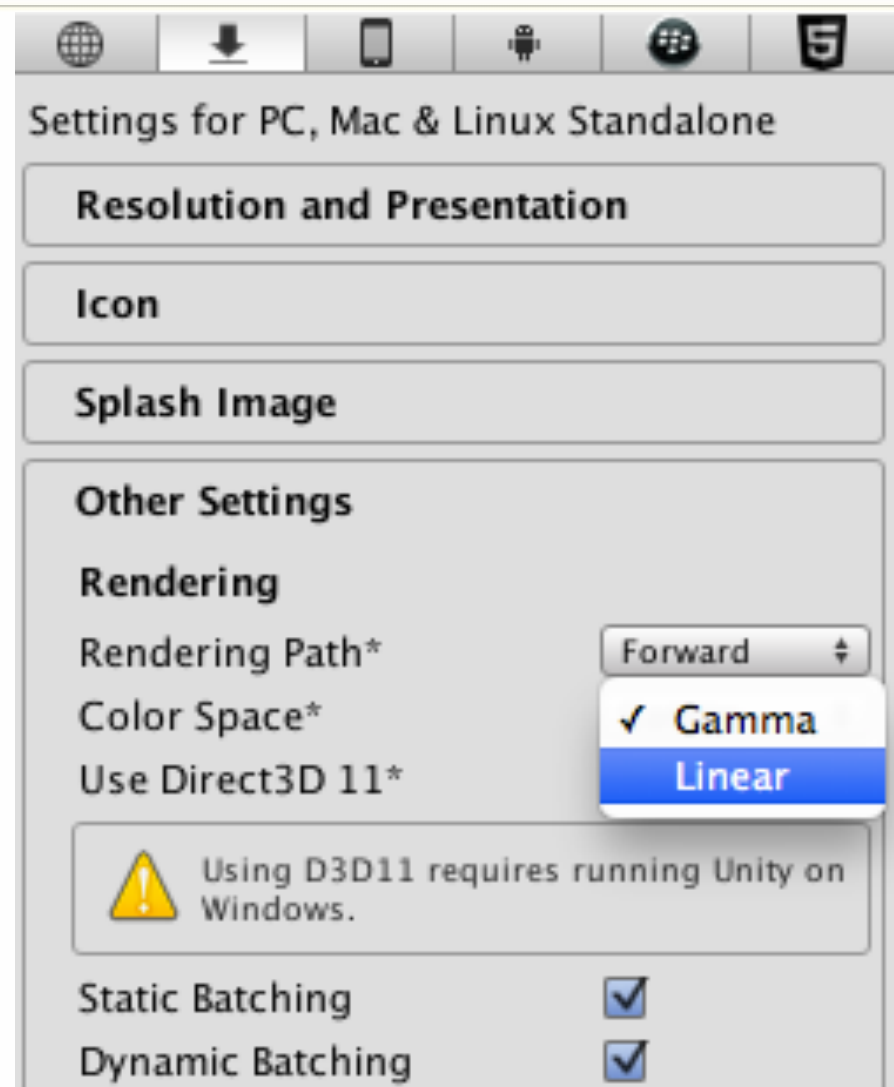
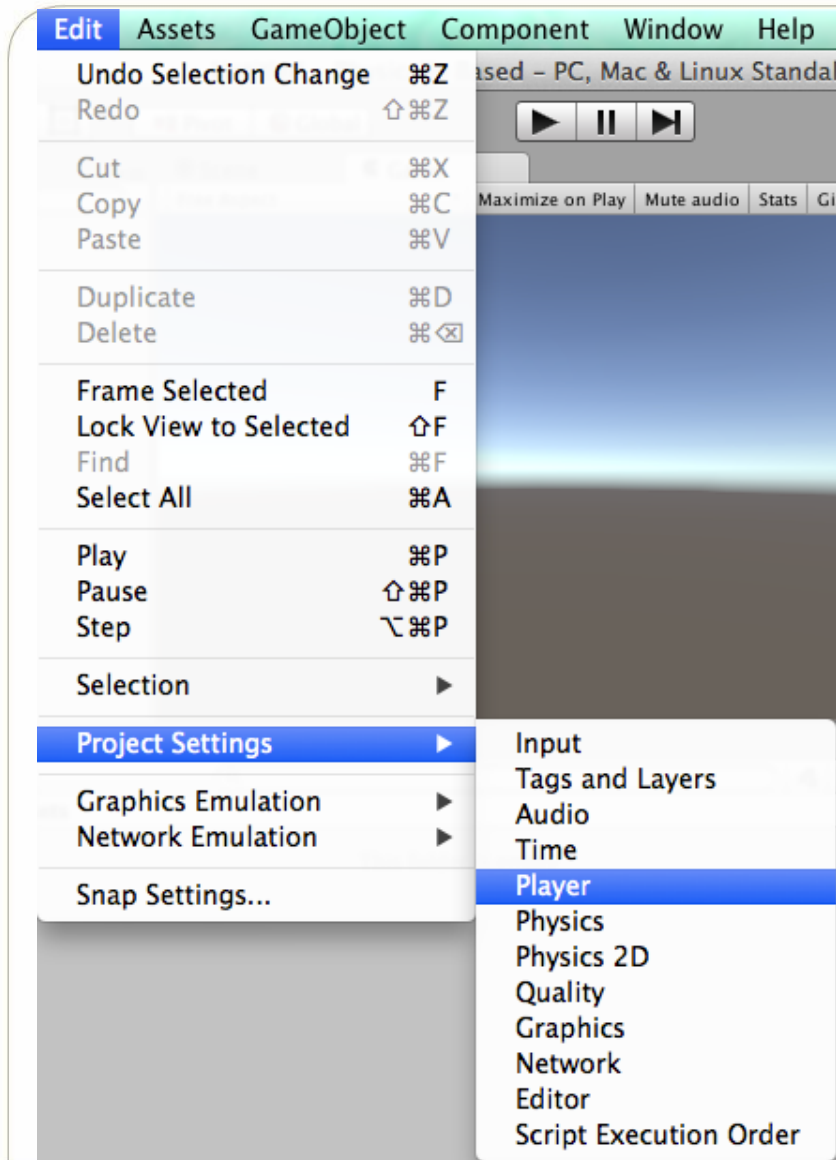
In your surface shader:

```
float4 tex = tex2D(_MainTex,IN.uv_MainTex);  
tex.rgb = DeGamma(tex.rgb);
```

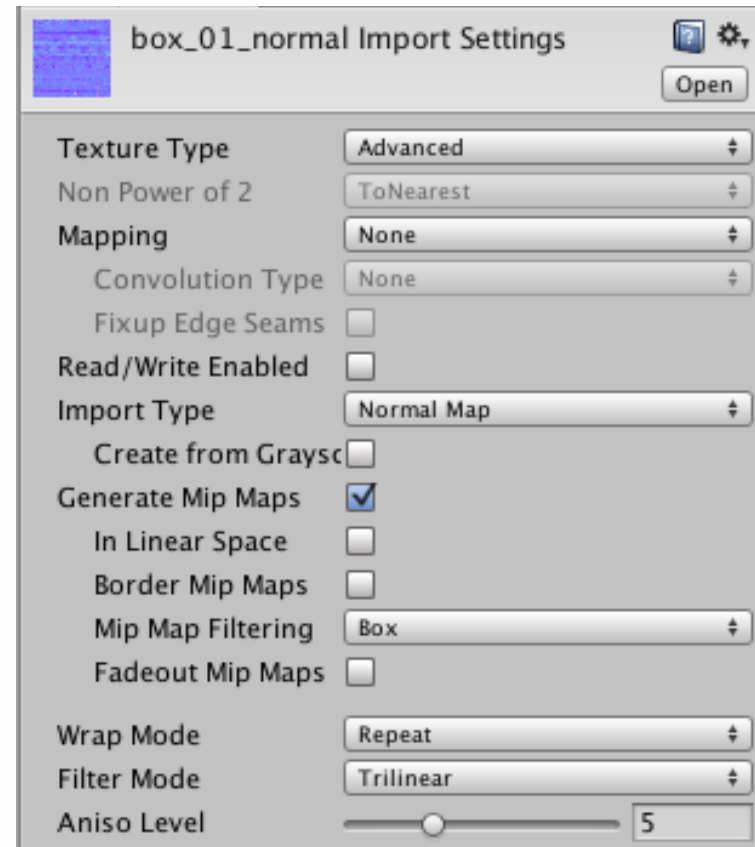
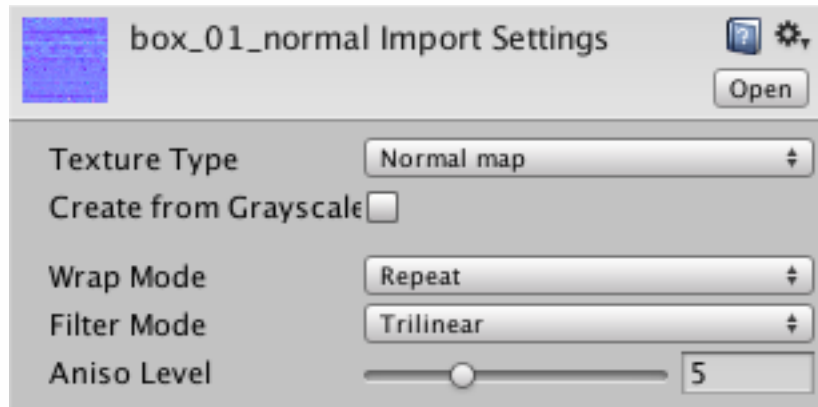
Let the GPU magically handle it

- Avoid wasting instructions in pixel shader
- Direct3D API:
 - Texture read: `D3DSAMP_SRGBTEXTURE`
 - Pixel write: `D3DRS_SRGBWRITEENABLE`
- OpenGL API:
 - Texture read: `GL_EXT_texture_sRGB`
 - Pixel write: `GL_EXT_framebuffer_sRGB`

Tell Unity to let GPU handle it

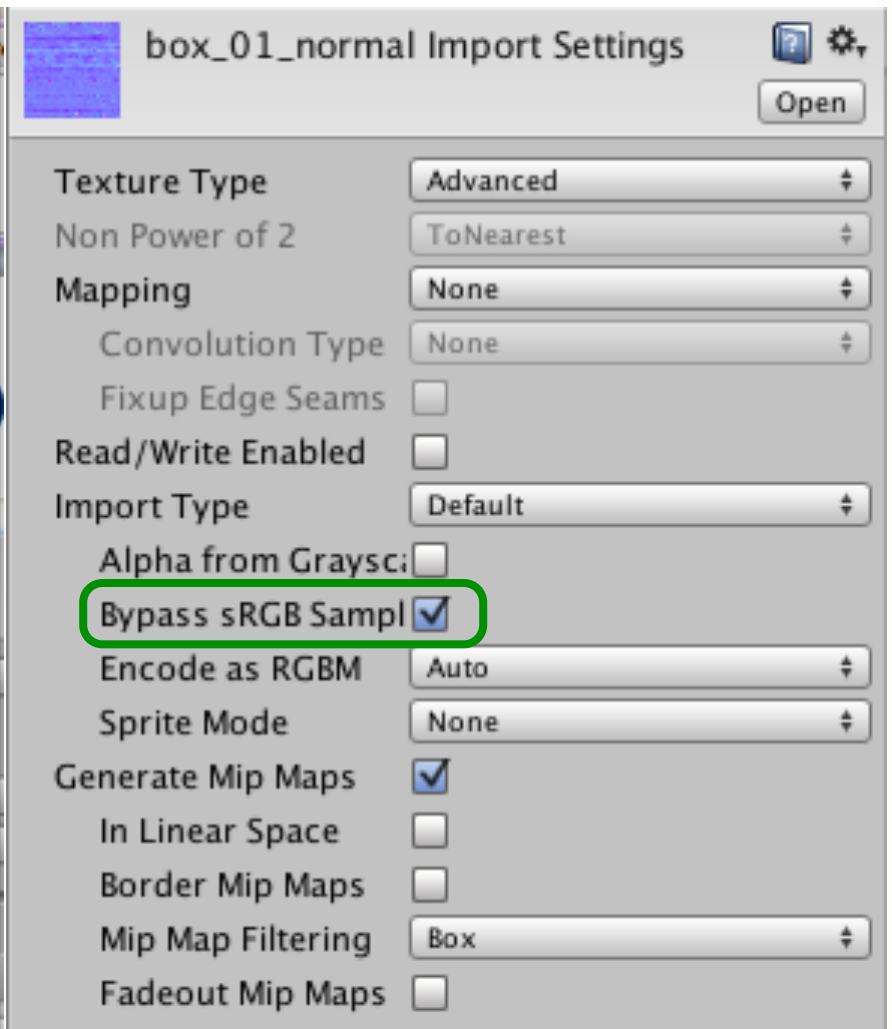
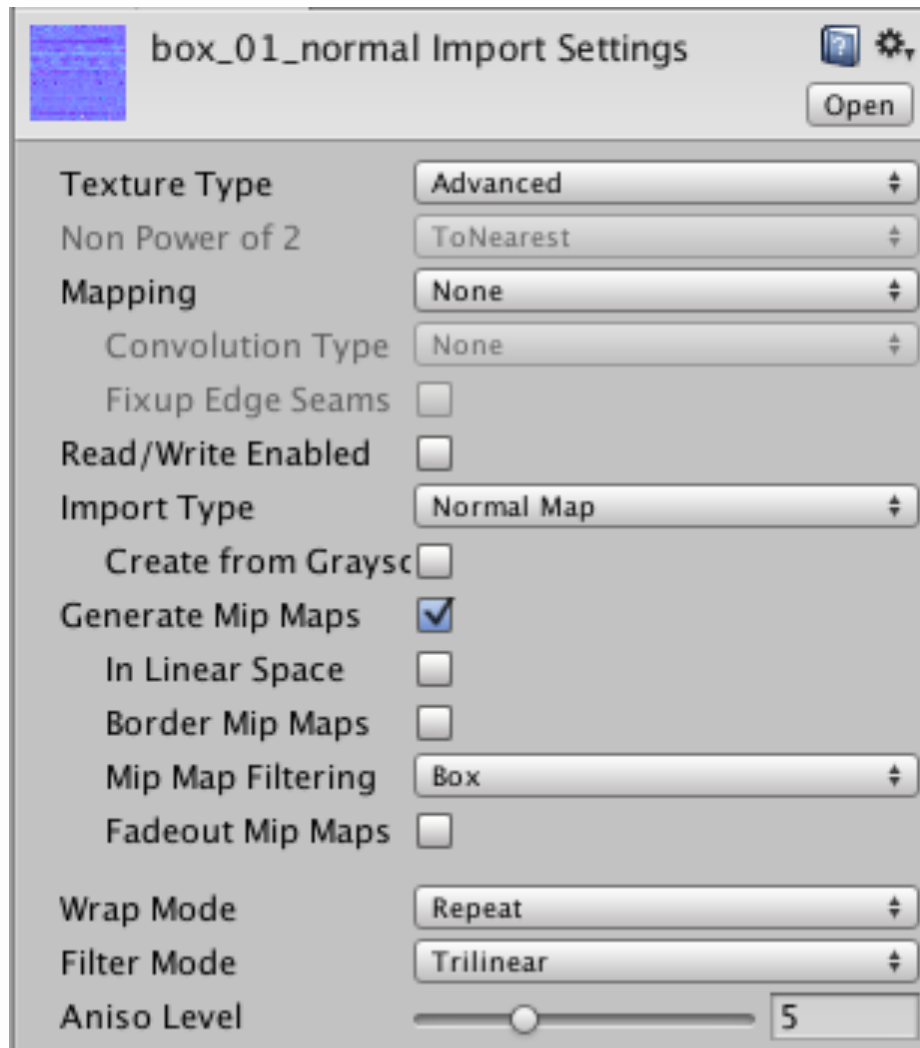


When to bypass sRGB sampling (1)



From http://http.developer.nvidia.com/GPUGems3/gpugems3_ch24.html

When to bypass sRGB sampling (2)



Bidirectional Reflectance Functions

- For “punctal” lights (typical point, directional, spotlight...)

$$C_{final} = C_{light} \otimes \pi BDRF(stuff) \max(0, n \cdot L)$$

- Classic diffuse lighting model: $BDRF = \frac{M_{diff}}{\pi}$

- Classic Blinn-Phong lighting model:

$$BRDF = \frac{M_{spec}}{\pi} \frac{(n \cdot H)^s}{n \cdot L} \text{ for } n \cdot L > 0$$

Reciprocity

- Reciprocity means we can swap the incoming and outgoing light directions in the BRDF
- Obviously true for classic diffuse lighting:

$$BRDF = \frac{M_{diff}}{\pi}$$

Reciprocity

- Does not hold for classic Blinn-Phong:

$$BRDF = \frac{M_{spec}}{\pi} \frac{\left(n \bullet \frac{L+V}{|L+V|} \right)^s}{n \bullet L} \quad \text{for } n \bullet L > 0$$

$$\neq \frac{M_{spec}}{\pi} \frac{\left(n \bullet \frac{V+L}{|V+L|} \right)^s}{n \bullet V} \quad \text{for } n \bullet V > 0$$

Modified Blinn-Phong

- Modified Blinn-Phong:

$$BRDF = \frac{M_{spec}}{\pi} (n \bullet H)^s$$

$$C_{final} = C_{light} \otimes \pi BDRF(stuff) \max(0, n \bullet L)$$

$$= C_{light} \otimes M_{spec} (n \bullet H)^s \max(0, n \bullet L)$$

Energy conservation

“Total amount of reflected light cannot be more than the amount of incoming light” – Rory Driscoll

- For classic diffuse model $BRDF = \frac{M_{diff}}{\pi}$

turns out you *technically* need $\pi M_{diff} < 1$

- Since π is just a constant, “we usually just ignore it and assume that our lights are just π times too bright.” – Steve McAuley

From <http://blog.stevemcauley.com/2011/12/03/energy-conserving-wrapped-diffuse>
and <http://www.rorydriscoll.com/2009/01/25/energy-conservation-in-games>

A common yet confusing convention

- For classic diffuse model $BRDF = \frac{M_{diff}}{\pi}$
we'll pretend you *practically* need $M_{diff} < 1$
- Convenient for artists: hit a diffuse surface with a material color of “1” with a directional light with an “intensity” of “1” parallel with its normal and you get “1”
- Will use same convention throughout
- If you are implementing global illumination, be careful!

Normalized modified Blinn-Phong

- To achieve energy conservation:

$$BRDF = \frac{s+8}{8\pi} M_{spec} (n \bullet H)^s$$

with *practical* need $M_{spec} < 1$

$$C_{final} = C_{light} \otimes \frac{s+8}{8} M_{spec} (n \bullet H)^s \max(0, n \bullet L)$$

From <http://blog.stevemcauley.com/2011/12/03/energy-conserving-wrapped-diffuse>
and <http://www.rorydriscoll.com/2009/01/25/energy-conservation-in-games>

Combining specular and diffuse

- Combined BRDF:

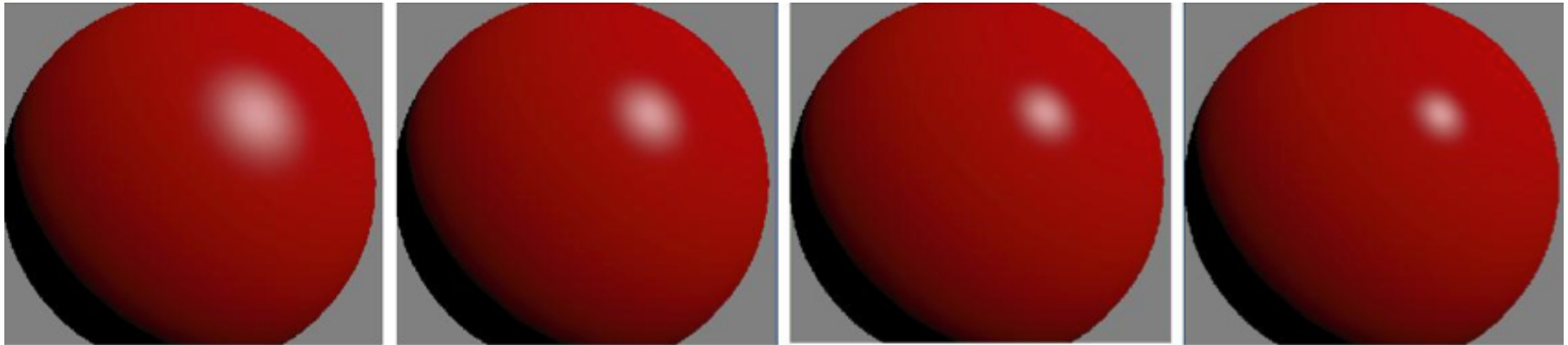
$$BRDF = \frac{M_{diff}}{\pi} + \frac{s+8}{8\pi} M_{spec} (n \bullet H)^s$$

with *practical* need $M_{diff} + M_{spec} < 1$

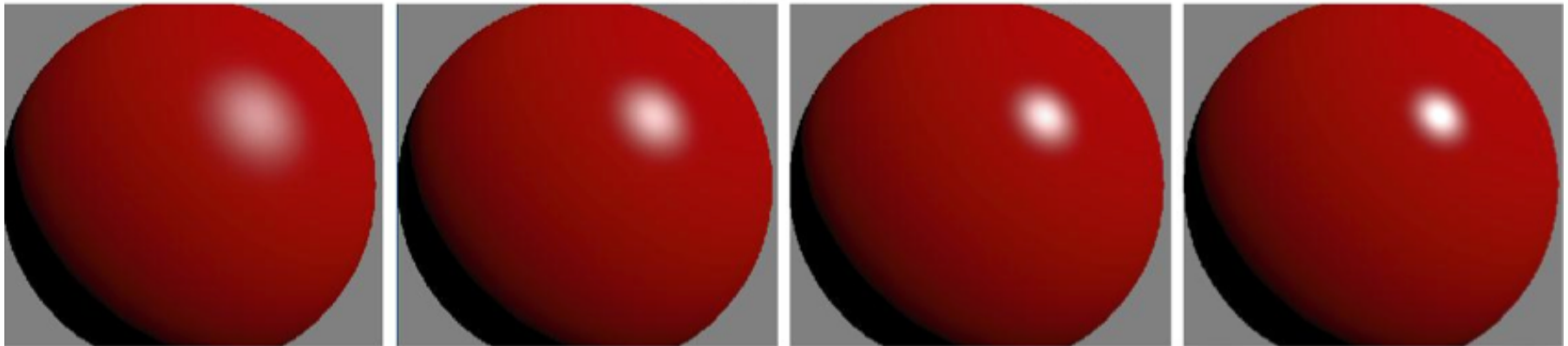
$$C_{final} = C_{light} \otimes \left[M_{diff} + \frac{s+8}{8} M_{spec} (n \bullet H)^s \right] \max(0, n \bullet L)$$

From <http://blog.stevemcauley.com/2011/12/03/energy-conserving-wrapped-diffuse>
and <http://www.rorydriscoll.com/2009/01/25/energy-conservation-in-games>

Normalized specular helps your artists



Without Normalization Factor



With Normalization Factor

Snapshoted from

http://renderwonk.com/publications/s2010-shading-course/hoffman/s2010_physically_based_shading_hoffman_b.pdf

Image from “Real-Time Rendering,” 3rd Edition, A K Peters 2008

Metals vs. dielectrics

- Dielectrics have diffuse and specular reflections
 - Specular reflection is white
- Metals have purely specular reflections
 - Specular reflection may be non-white
 - Still white at large grazing angles

Of course, metal plates are shiny



Diffuse



Specular

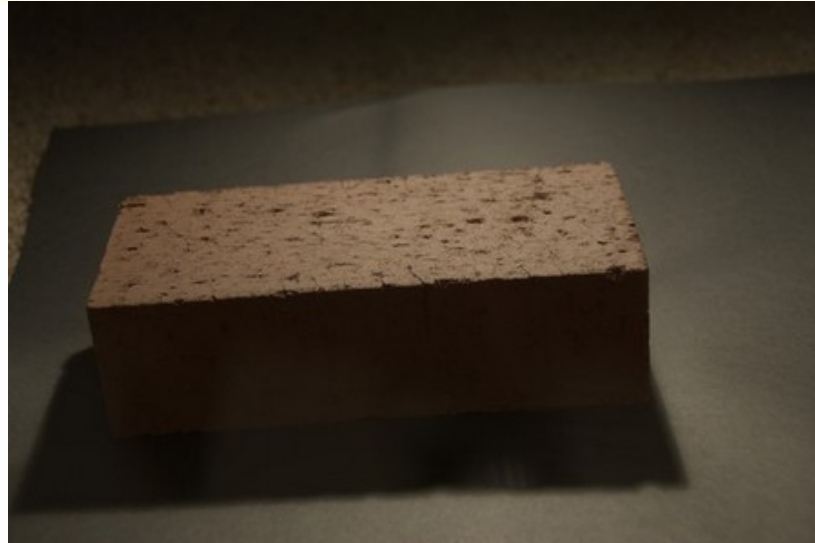


From <http://filmicgames.com/archives/547>

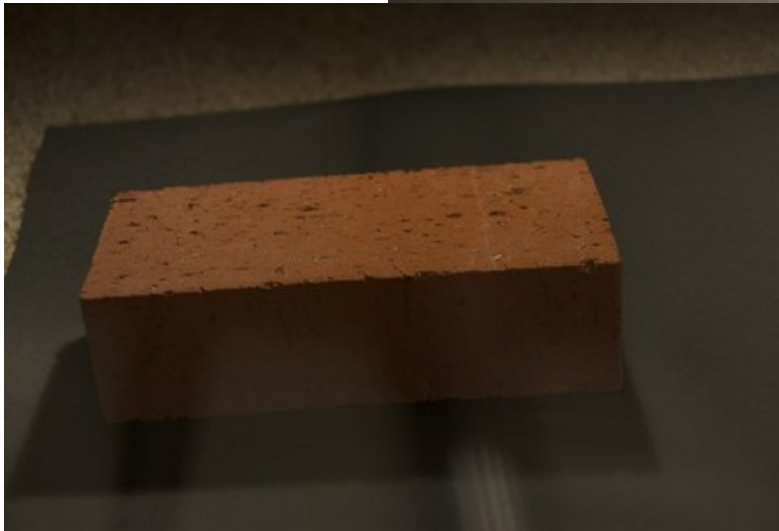
But bricks are shiny too



Diffuse



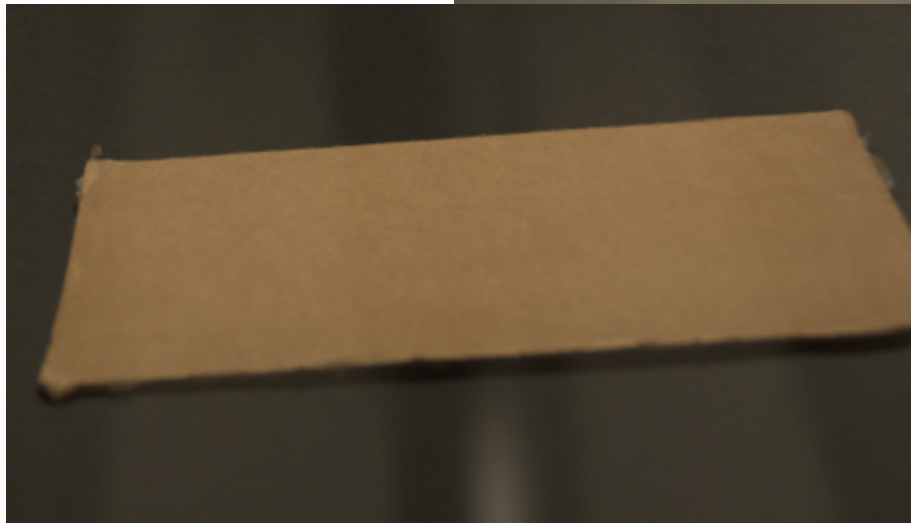
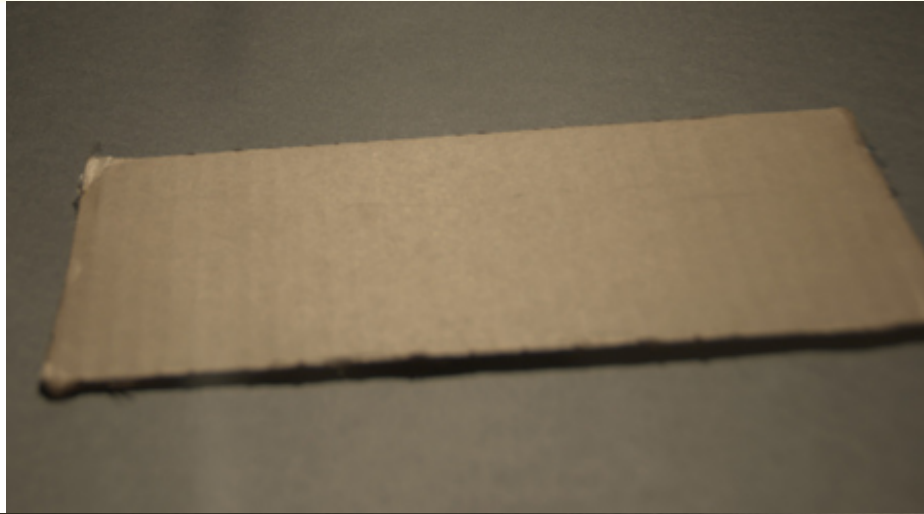
Specular



From <http://filmicgames.com/archives/547>

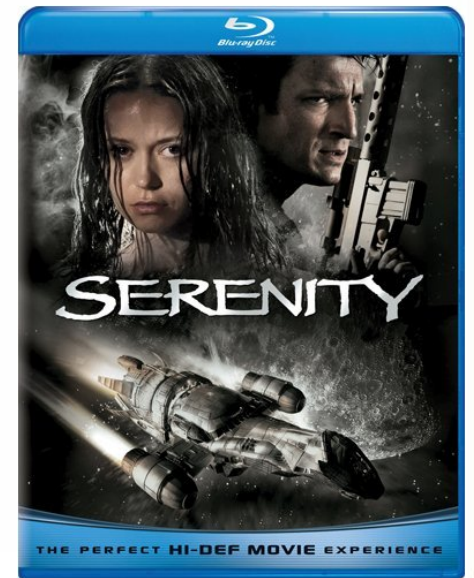
**But cardboard
can't be shiny,
right?**

Nope, cardboard is shiny!



From <http://filmicgames.com/archives/547>

**BECAUSE
EVERYTHING
IS
SHINY!**



Pictures from amazon.com



John Hable on cardboard

“Poor cardboard.

So misunderstood.

It always gets referred to as a ‘pure diffuse material’, even though it actually deserves to hang out with its shiny friends.”

– John Hable

Fresnel effect

- Specular reflections increase dramatically at high grazing angles
- Schlick approximation:

$$F(V, H) = F_0 + (1 - F_0)(1 - V \cdot H)^5$$

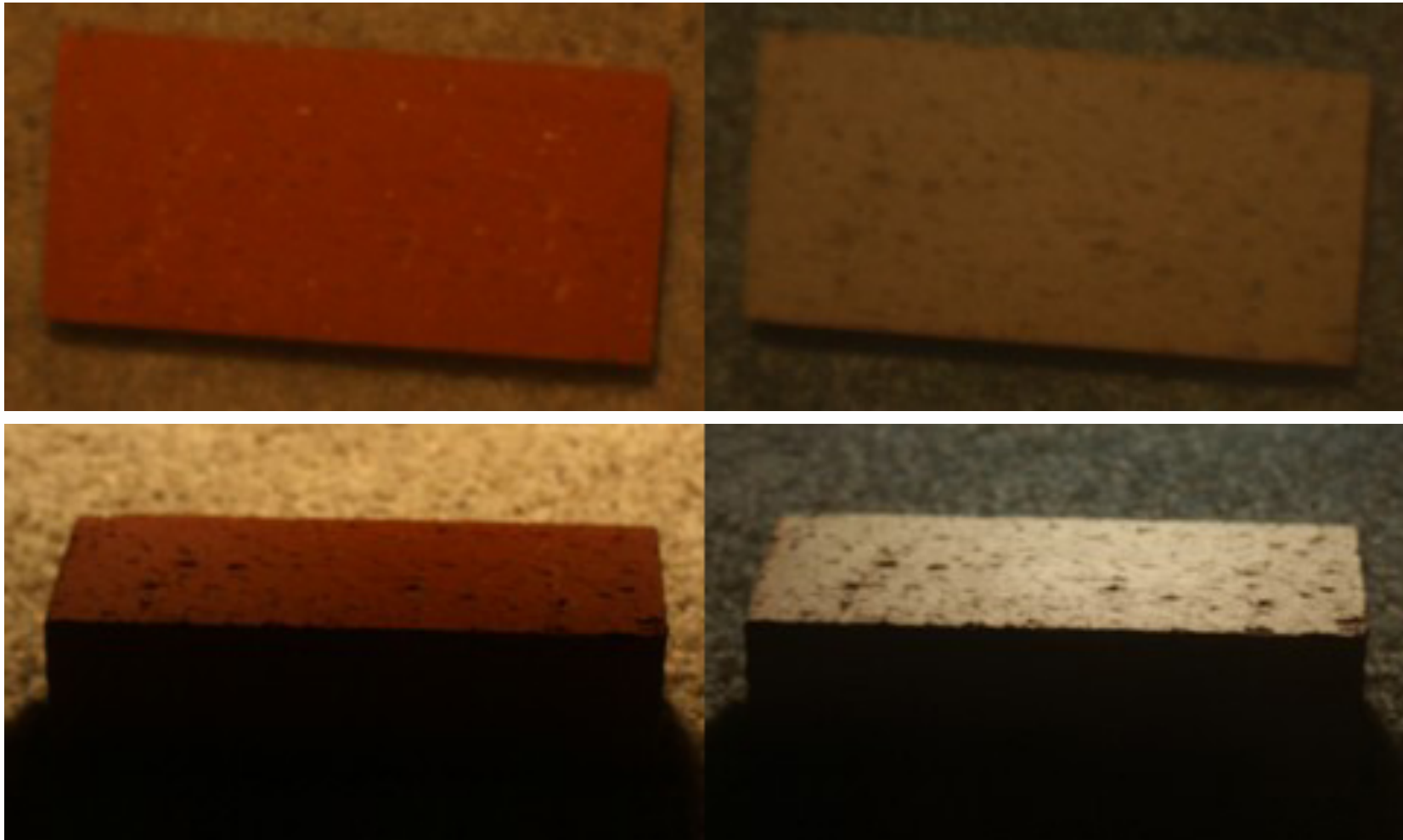
$$= F_0 + (1 - F_0)(1 - L \cdot H)^5$$

Reflection when view
and light vector align

white

(grey for dielectrics, but metals can have color)

Bricks have Fresnel



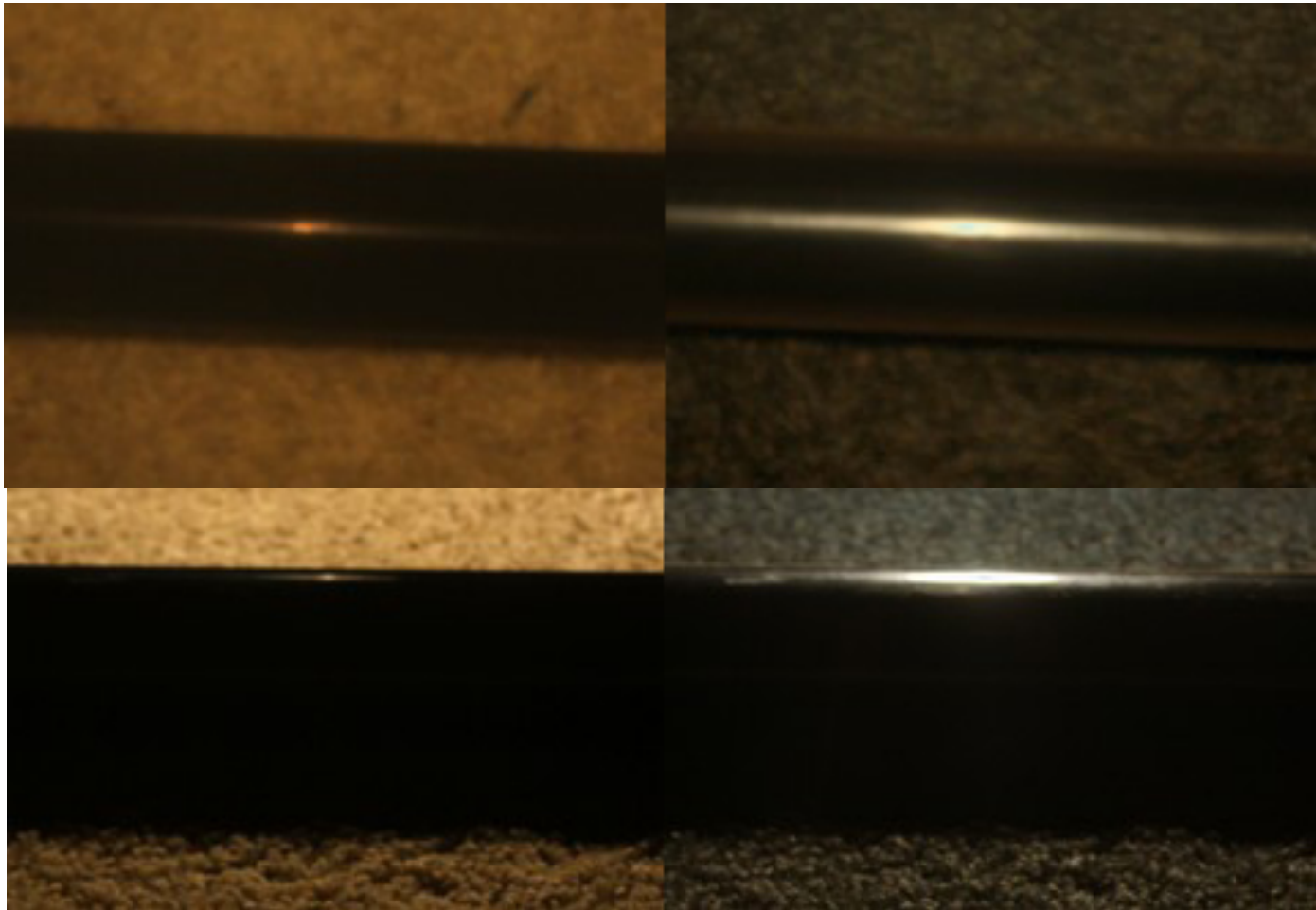
From <http://filmicgames.com/archives/557>

Cardboard has Fresnel



From <http://filmicgames.com/archives/557>

PVC pipes have Fresnel



From <http://filmicgames.com/archives/557>

EVERYTHING HAS FRESNEL



Pictures from wikipedia

F_0 for dielectrics (linear)

Quartz	0.045593921
Ice	0.017908907
Water	0.020373188
Alcohol	0.01995505
Glass	0.04
Milk	0.022181983
Ruby	0.077271957
Crystal	0.111111111
Diamond	0.171968833
Skin	0.028

**Remember
specular
reflection of
dielectrics
is white!**

From <https://seblagarde.wordpress.com/2011/08/17/feeding-a-physical-based-lighting-mode/>

F_0 for metals (linear)

	R	G	B
Silver	0.971519	0.959915	0.915324
Aluminium	0.913183	0.921494	0.924524
Gold	1	0.765557	0.336057
Copper	0.955008	0.637427	0.538163
Chromium	0.549585	0.556114	0.554256
Nickel	0.659777	0.608679	0.525649
Titanium	0.541931	0.496791	0.449419
Cobalt	0.662124	0.654864	0.633732
Platinum	0.672411	0.637331	0.585456

From <https://seblagarde.wordpress.com/2011/08/17/feeding-a-physical-based-lighting-mode/>

Incorporating Fresnel term

- Combined BRDF:

$$BRDF = \frac{M_{diff}}{\pi} + \frac{s+8}{8\pi} F(V, H) (n \bullet H)^s$$

with *practical* need $M_{diff} + F(V, H) < 1$

$$C_{final} = C_{light} \otimes \left[M_{diff} + \frac{s+8}{8} F(v, h) (n \bullet H)^s \right] \max(0, n \bullet L)$$

Top equation corresponds to equation
7.49 on p. 257 of “Real-Time Rendering,”
3rd Edition, A K Peters 2008

A reflective paradox?



From <http://www.thetenthplanet.de/archives/255>

General Cook-Torrance specular

- Combined BRDF:

$$BRDF = \frac{M_{diff}}{\pi} + \frac{D(n \bullet H)F(V \bullet H)G(n, L, V)}{4(n \bullet L)(n \bullet V)}$$

Blinn-Phong microfacet distribution

- Microfacet Blinn-Phong:

$$D_{BP}(n \cdot H) = \frac{s+2}{2\pi} (n \cdot H)^s$$

- Throughout rest of slides, imagine the artist makes “smoothness maps” containing “gloss” g in the range $[0,1]$
- Call of Duty: Black Ops

$$s = 8192^g = 2^{(13g)}$$

GGX microfacet distribution

- GGX:

$$D_{GGX}(n \bullet H) = \frac{\alpha^2}{\pi[(n \bullet H)^2(\alpha^2 - 1) + 1]^2}$$

- Crytek: $\alpha = (1 - 0.7g)^2$
- Unreal: $\alpha = (\text{disney roughness})^2 = (1 - g)^2$

Original Cook-Torrance geometric term

- From original Cook-Torrance paper:

$$G_{CK}(n, L, V) = \min\left(1, \frac{2(n \cdot H)(n \cdot V)}{V \cdot H}, \frac{2(n \cdot H)(n \cdot L)}{L \cdot H}\right)$$

GGX-based geometric term (1)

- GGX-based geometric:

$$G_{GGX-R}(n, L, V) = G_{GGX-R}(n \bullet L) G_{GGX-R}(n \bullet V)$$

$$G_{GGX-R}(n \bullet V) = \frac{2(n \bullet V)}{n \bullet V + \sqrt{\alpha^2 + (1 - \alpha^2)(n \bullet V)^2}}$$

- Used in The Order: 1886

GGX-based geometric term (2)

- Ideal of a “vizualization” term:

$$Viz(n, L, V) = \frac{G(n, L, V)}{4(n \bullet L)(n \bullet V)}$$

$$Viz_{GGX-R}(n, L, V) = \frac{1}{n \bullet V + \sqrt{\alpha^2 + (1 - \alpha^2)(n \bullet V)^2}} \\ \times \frac{1}{n \bullet L + \sqrt{\alpha^2 + (1 - \alpha^2)(n \bullet L)^2}}$$

GGX-matched to GGX geometric term

- GGX-form matched to GGX distribution:

$$G_{GGX-M}(n, L, V) = G_{GGX-M}(n \bullet L) G_{GGX-M}(n \bullet V)$$

$$G_{GGX-M}(n \bullet V) = \frac{n \bullet V}{(n \bullet V)(1 - k) + k}$$

- Disney standard: $k = \alpha / 2 = (g - 1)^2 / 2$
- Disney “hotness remapping”:

$$k = [0.5 + (1 - g) / 2]^2 / 2$$

Implicit geometric term

- Implicit geometric:

$$G_I(n, L, V) = 4(n \bullet L)(n \bullet V)$$

$$Viz_I(n, L, V) = \frac{G_I(n, L, V)}{4(n \bullet L)(n \bullet V)} = 1$$

Keleman geometric term

- Facilitates fast implementation:

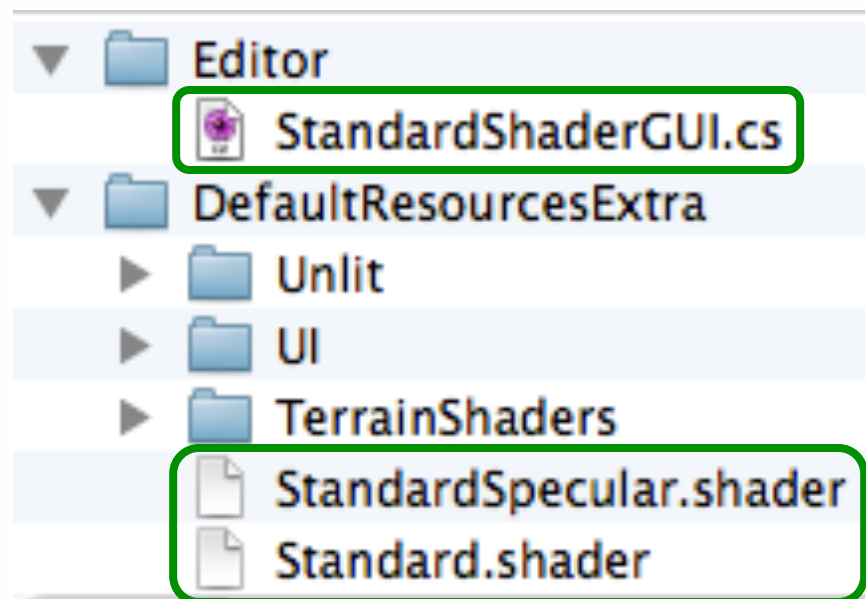
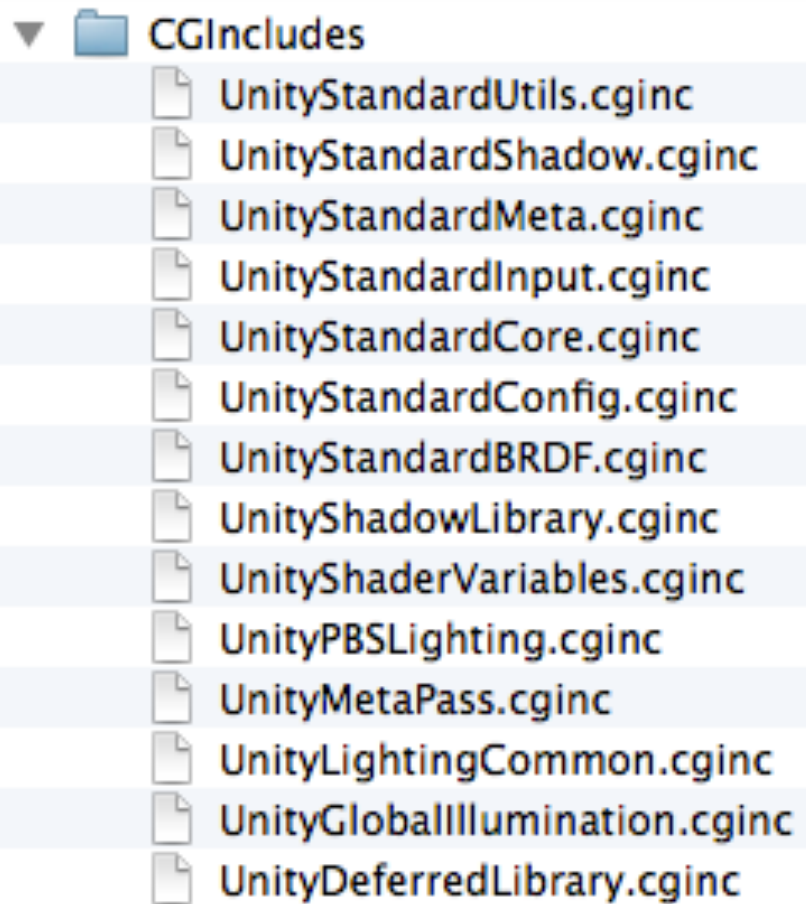
$$G_K(n, L, V) = \frac{(n \bullet L)(n \bullet V)}{4(V \bullet H)^2}$$

$$Viz_K(n, L, V) = \frac{1}{4(V \bullet H)^2}$$

Unity 5's “Standard Shader”

- Source available separately:

<http://unity3d.com/unity/download/archive>



SurfaceOutputStandard structure

- Your job is to fill this in, as necessary:

```
struct SurfaceOutputStandard {  
    fixed3 Albedo;           // base (diffuse or specular) color  
    fixed3 Normal;           // tangent space normal, if written  
    half3 Emission;  
    half Metallic;           // 0=non-metal, 1=metal  
    half Smoothness;         // 0=rough, 1=smooth  
    half Occlusion;          // occlusion (default 1)  
    fixed Alpha;             // alpha for transparencies  
};
```

From <http://docs.unity3d.com/Manual/SL-SurfaceShaders.html>

SurfaceOutputStandardSpecular structure

- Your job is to fill this in, as necessary:

```
struct SurfaceOutputStandardSpecular {  
    fixed3 Albedo;           // diffuse color  
    fixed3 Specular;         // specular color  
    fixed3 Normal;           // tangent space normal, if written  
    half3 Emission;  
    half Smoothness;         // 0=rough, 1=smooth  
    half Occlusion;           // occlusion (default 1)  
    fixed Alpha;             // alpha for transparencies  
};
```

From <http://docs.unity3d.com/Manual/SL-SurfaceShaders.html>