SIGGRAPH 2020 S2020.SIGGRAPH.ORG THINK BEYOND

HIGH QUALITY, HIGH PERFORMANCE GRAPHICS IN FILAMENT

Romain Guy

What is Filament?

https://github.com/google/filament

Design goals

Physically based rendering in Filament

https://google.github.io/filament/Filament.html

4.7.2 Energy loss in specular reflectance

The Cook-Torrance BRDF we presented earlier attempts to model several events at the microfacet level but does so by accounting for a single bounce of light. This approximation can cause a loss of energy at high roughness, the surface is not energy preserving. Figure 12 shows why this loss of energy occurs. In the single bounce (or single scattering) model, a ray of light hitting the surface can be reflected back onto another microfacet and thus be discarded because of the masking and shadowing term. If we however account for multiple bounces (multiscattering), the same ray of light might escape the microfacet field and be reflected back towards the viewer.

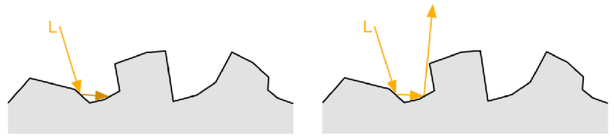


Figure 12: *Single scattering (left) vs multiscattering*

Based on this simple explanation, we can intuitively deduce that the rougher a surface is, the higher the chances are that energy gets lost because of the failure to account for multiple scattering events. This loss of energy appears to darken rough materials. Metallic surfaces are particularly affected because all of their reflectance is specular. This darkening effect is illustrated in figure 13. With multiscattering, energy preservation can be achieved, as shown in figure 14.

Figure 13: *Darkening increases with roughness due to single scattering*

Figure 14: *Energy preservation with multiscattering*

We can use a white furnace, a uniform lighting environment set to pure white, to validate the energy preservation property of a BRDF. When energy preservation is achieved, a purely reflective metallic surface ($f_0 = 1$) should be indistinguishable from the background, no matter the roughness of said surface. Figure 15 shows what such a surface looks like with the specular BRDF presented in the previous sections. The loss of energy as the roughness increases is obvious. In contrast, figure 16 shows that accounting for multiscattering events addresses the energy loss.

Figure 15: *Darkening increases with roughness due to single scattering*

Figure 16: *Energy preservation with multiscattering*

Multiple-scattering microfacet BRDFs are discussed in depth in [Heitz16]. Unfortunately this paper only presents a stochastic evaluation of the multiscattering BRDF. This solution is therefore not suitable for real-time rendering. Kulla and Conty present a different approach in [Kulla17]. Their idea is to add an energy compensation term as an additional BRDF lobe shown in equation 23:

$$f_{ms}(l, v) = \frac{(1 - E(l))(1 - E(v))F_{avg}^2 E_{avg}}{\pi(1 - E_{avg})(1 - F_{avg}(1 - E_{avg}))} \tag{23}$$

Where E is the directional albedo of the specular BRDF f_r, with f_0 set to 1:

$$E(l) = \int_\Omega f(l, v)(n \cdot v)dv \tag{24}$$

The term E_{avg} is the cosine-weighted average of E:

$$E_{avg} = 2 \int_0^1 E(\mu)\mu d\mu \tag{25}$$

Optimizing for tilers

resolve/unresolve

glInvalidateFramebuffer()

before draw calls → prevent unresolves

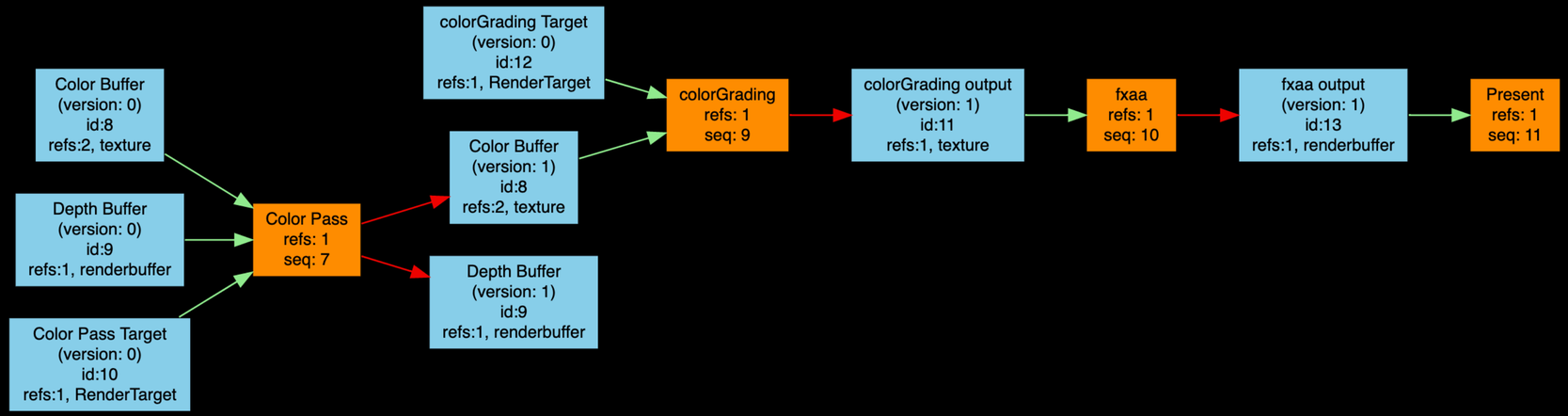
after draw calls → prevent resolves

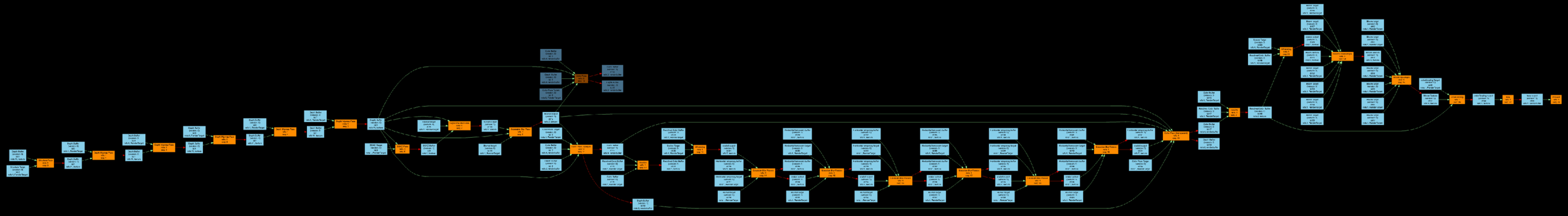
Framegraph

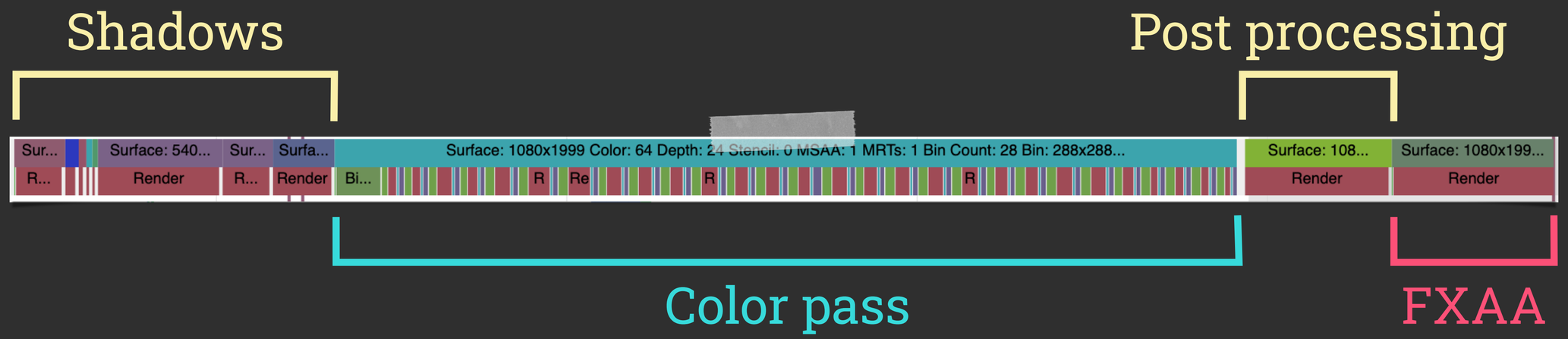
knows all buffer dependencies

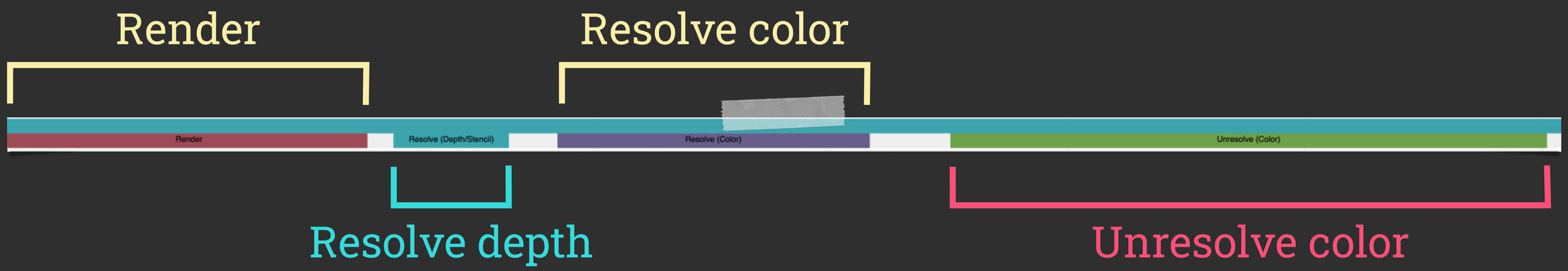
manages buffers lifetime

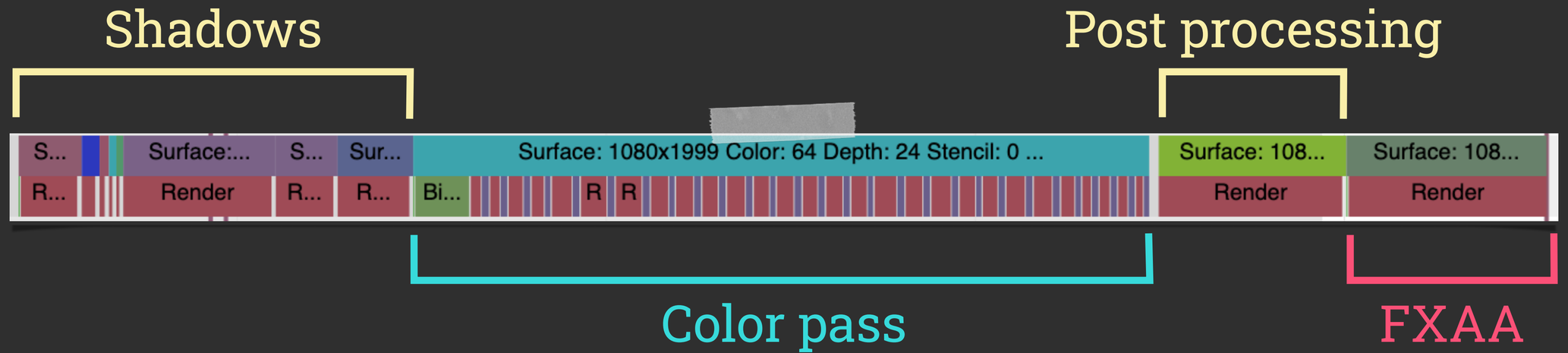
computes discard flags

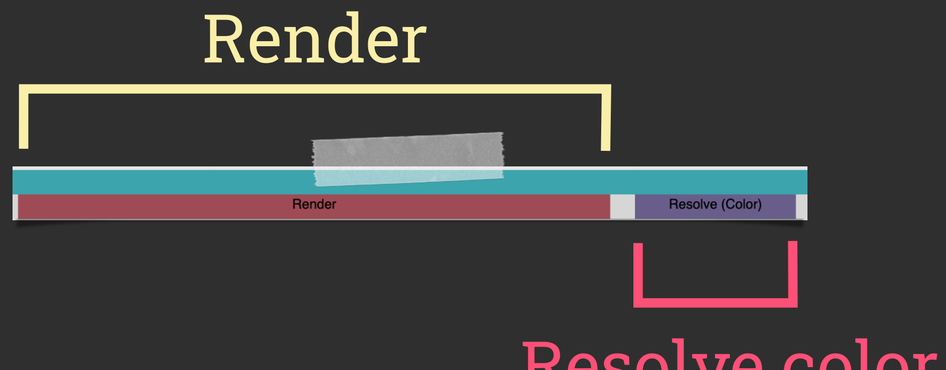

Color Buffer
(version: 0)
id:8
refs:2, texture
Depth Buffer
(version: 0)
id:9
refs:1, renderbuffer
Color Pass Target
(version: 0)
id:10
refs:1, RenderTarget
Color Pass
refs: 1
seq: 7
colorGrading Target
(version: 0)
id:12
refs:1, RenderTarget
Color Buffer
(version: 1)
id:8
refs:2, texture
Depth Buffer
(version: 1)
id:9
refs:1, renderbuffer
colorGrading
refs: 1
seq: 9
colorGrading output
(version: 1)
id:11
refs:1, texture
fxaa
refs: 1
seq: 10
fxaa output
(version: 1)
id:13
refs:1, renderbuffer
Present
refs: 1
seq: 11




Shadows
Post processing
Sur...
Surface: 540...
Sur...
Surfa...
Surface: 1080x1999 Color: 64 Depth: 24 Stencil: 0 MSAA: 1 MRTs: 1 Bin Count: 28 Bin: 288x288...
Surface: 108...
Surface: 1080x199...
R...
Render
R...
Render
Bi...
R
Re
R
R
Render
Render
Color pass
FXAA


Render
Resolve color
Render
Resolve (Depth/Stencil)
Resolve (Color)
Unresolve (Color)
Resolve depth
Unresolve color


Shadows
Post processing
S...
Surface:...
S...
Sur...
Surface: 1080x1999 Color: 64 Depth: 24 Stencil: 0 ...
Surface: 108...
Surface: 108...
R...
Render
R...
R...
Bi...
R
R
Render
Render
Color pass
FXAA


Render
Render
Resolve (Color)
Resolve color

We can do better

Post-processing

(sometimes) only needs current fragment

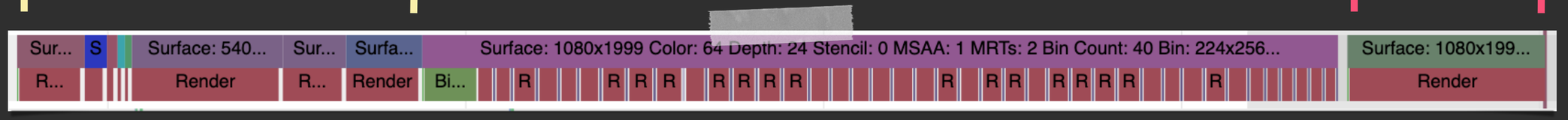
perfect candidate for
Vulkan subpasses/GL_EXT_shader_framebuffer_fetch

Recipe

1. Use 2 MRTs: 1 HDR (RGB16F), 1 LDR (RGB8)
2. Render color pass into HDR target
3. Render full-screen triangle into LDR target
    3.1. Shader fetches from HDR target
    3.2. Apply color grading, tone mapping, dithering, etc.
    3.3. Outputs to LDR target

Shadows

FXAA



Color pass + post processing

fp16

```
precision mediump float;

out vec4 fragColor;

void main() {
    // Do work
}
```

It can’t be that easy

min value → 0.000061035

max value → 65,504

precision in [1,024..2,048] → 1

```
// we use x / (roughness^4) in our code
// min value in fp16 → 2^-14 = 0.000061035
// we want roughness^4 >= 2^-14
// so min roughness = 2^(-14/4) ~= 0.089
roughness = clamp(roughness, 0.089, 1.0);
```

```
float D_GGX_Anisotropic(float at, float ab, float ToH, float BoH, float NoH) {
    // at & ab → roughness^2, a2 → roughness^4
    // The dot product below computes roughness^8
    // → won't fit in fp16 without clamping the roughness to 0.298
    // → perform the dot product and the division in fp32
    float a2 = at * ab;

    highp vec3 d = vec3(ab * ToH, at * BoH, a2 * NoH);
    highp float d2 = dot(d, d);

    float b2 = a2 / d2;
    return a2 * b2 * b2 * (1.0 / PI);
}
```

```
#define MEDIUMP_FLT_MAX        65504.0
#define saturateMediump(x)     min(x, MEDIUMP_FLT_MAX)

float V_SmithGGXCorrelated(float roughness, float NoV, float NoL) {
    float v = // ...
    return saturateMediump(v);
}
```

```
// GPU computation
color = lightIntensity * cameraExposure * BRDF();
```

> max fp16 (65,504)

```
// Intensity of the Sun in lux
float lightIntensity = 110000.0f;

// Exposure for a clear day:
//   ƒ/16, 1/125s, ISO 100
float cameraExposure = 0.000026042;
```

< min fp16 (0.000061035)

```
// CPU computation, using 32-bit floats
float preExposedLightIntensity =
    cameraExposure * lightIntensity;

// GPU computation, using 16-bit floats
color = preExposedLightIntensity * BRDF();
```

2.86462

Be careful

Faster* BRDFs

$$V(v,l,\alpha) = \frac{0.5}{n \cdot l\sqrt{(n \cdot v)^2(1-\alpha^2)+\alpha^2} + n \cdot v\sqrt{(n \cdot l)^2(1-\alpha^2)+\alpha^2}}$$

$$\sqrt{a^2b^2 + c^2} \approx ab + c$$

$$V(v,l,\alpha) = \frac{0.5}{n \cdot l(n \cdot v(1-\alpha)+\alpha) + n \cdot v(n \cdot l(1-\alpha)+\alpha)}$$

$$V(v,l,\alpha) = \frac{0.5}{2(n \cdot l)(n \cdot v)(1-\alpha) + (n \cdot v + n \cdot l)\alpha}$$

```
float V_SmithGGXCorrelated_Fast(float roughness, float NoV, float NoL) {
    // Hammon 2017, "PBR Diffuse Lighting for GGX+Smith Microsurfaces"
    return 0.5 / mix(2.0 * NoL * NoV, NoL + NoV, roughness);
}
```

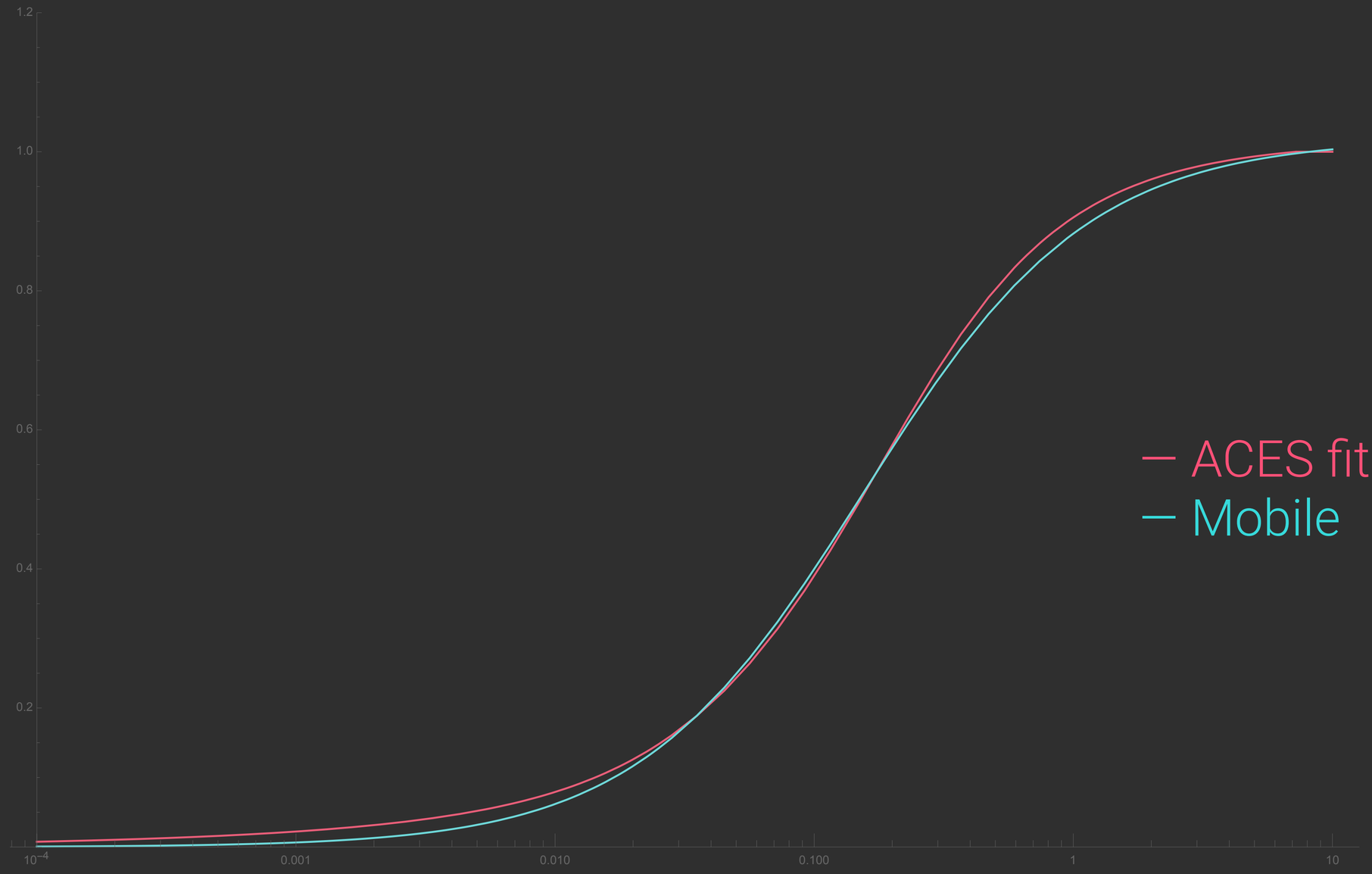
$$V_{clearCoat}(l, h) = \frac{1}{4(l \cdot h)^2}$$

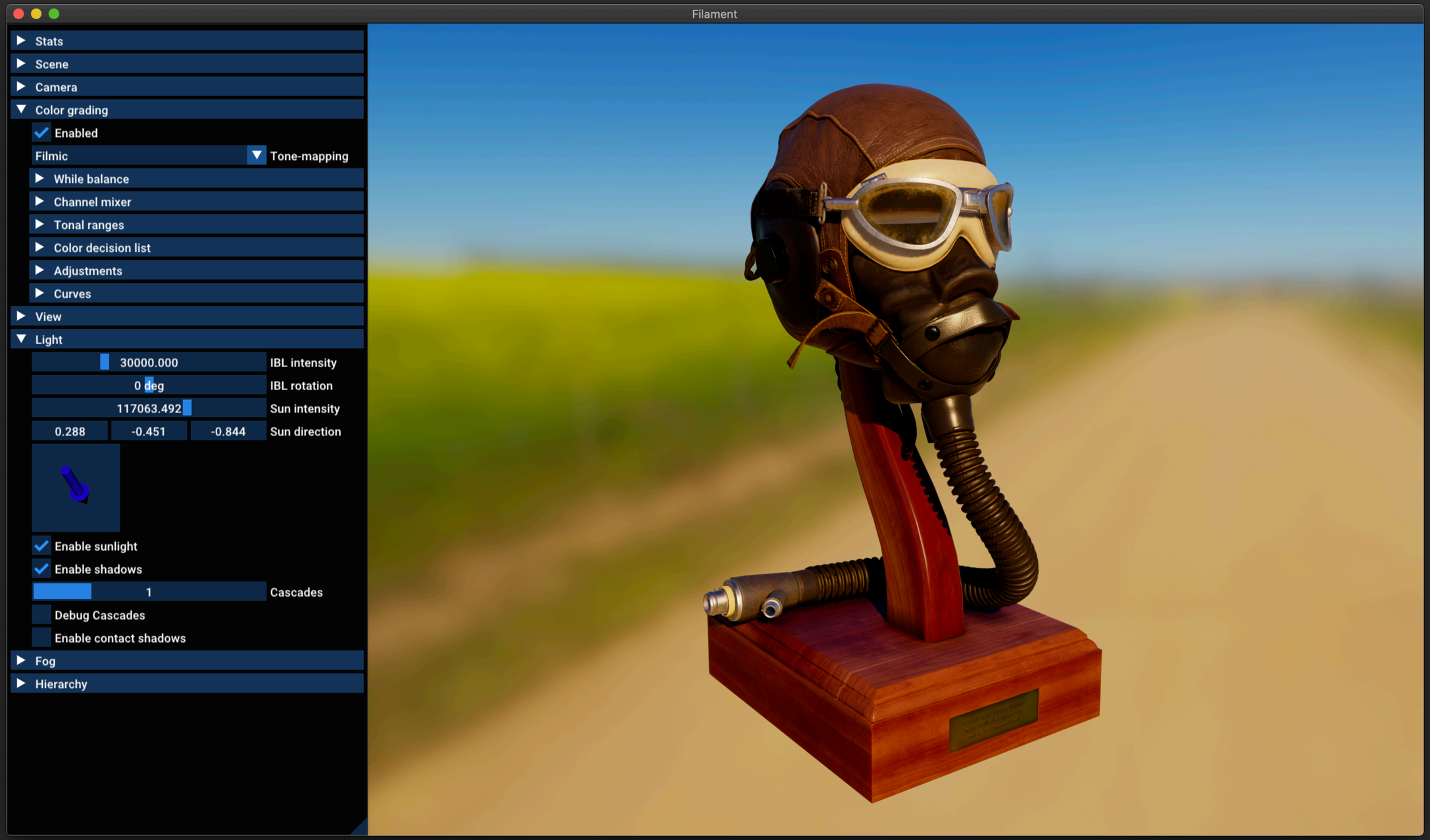
Tone mapping

Krzysztof Narkowicz has a popular fit of ACES

```
vec3 Tonemap_ACES(const vec3 x) {
    // Narkowicz 2015, "ACES Filmic Tone Mapping Curve”
    const float a = 2.51;
    const float b = 0.03;
    const float c = 2.43;
    const float d = 0.59;
    const float e = 0.14;
    return (x * (a * x + b)) / (x * (c * x + d) + e);
}
```

```
vec3 Tonemap_Mobile(const vec3 x) {
    // Transfer function baked in,
    // don't use with sRGB OETF!
    return x / (x + 0.155) * 1.019;
}
```


1.2
1.0
0.8
0.6
0.4
0.2
10^{-4}
0.001
0.010
0.100
1
10
ACES fit
Mobile


Filament
Stats
Scene
Camera
Color grading
Enabled
Filmic
Tone-mapping
While balance
Channel mixer
Tonal ranges
Color decision list
Adjustments
Curves
View
Light
30000.000
IBL intensity
0 deg
IBL rotation
117063.492
Sun intensity
0.288
-0.451
-0.844
Sun direction
Enable sunlight
Enable shadows
1
Cascades
Debug Cascades
Enable contact shadows
Fog
Hierarchy


Filament
Stats
Scene
Camera
Color grading
Enabled
Filmic
Tone-mapping
While balance
Channel mixer
Tonal ranges
Color decision list
Adjustments
Curves
View
Light
30000.000
IBL intensity
0 deg
IBL rotation
117063.492
Sun intensity
0.288
-0.451
-0.844
Sun direction
Enable sunlight
Enable shadows
1
Cascades
Debug Cascades
Enable contact shadows
Fog
Hierarchy

Image-based lighting

Spherical harmonics

SH 2 or 3 bands for diffuse lighting

easy to compute offline & cheap to apply

looks great for low/medium dynamic range environments

Diffuse map?

costs a sampler

more memory & storage

not worth it for us

Specular map!

diffuse lighting ~ specular lighting at roughness=1

$$\sum L(l) \approx \sum \langle n \cdot l \rangle L(h)$$

roughness 1.0 → cannot be 1x1 LOD

looks good enough

Roughness mapping

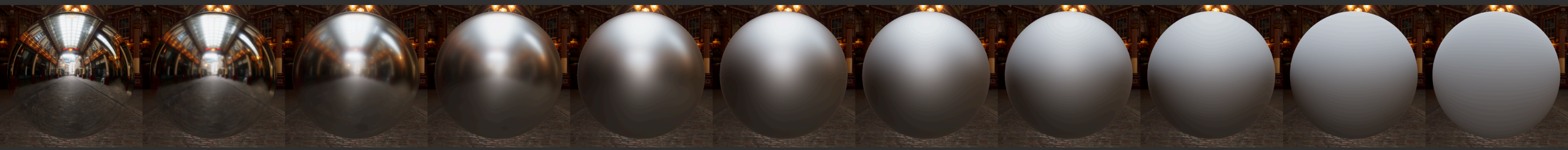
256x256 to 16x16 → 5 mip levels

$$log_2(roughness) + roughnessOneLOD$$

$$roughnessOneLOD * roughness * (2 - roughness)$$

Roughness mapping

0	→	0.000
1	→	0.018
2	→	0.086
3	→	0.250
4	→	1.000

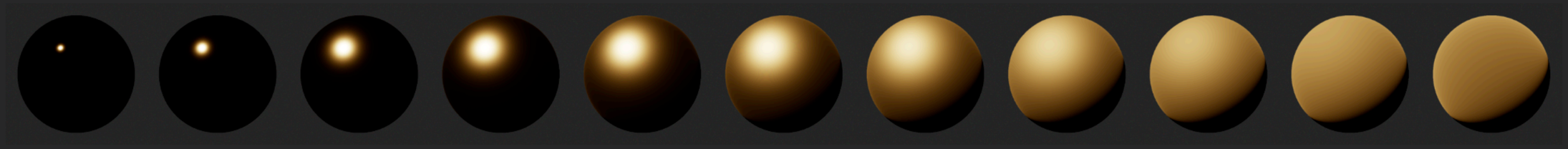


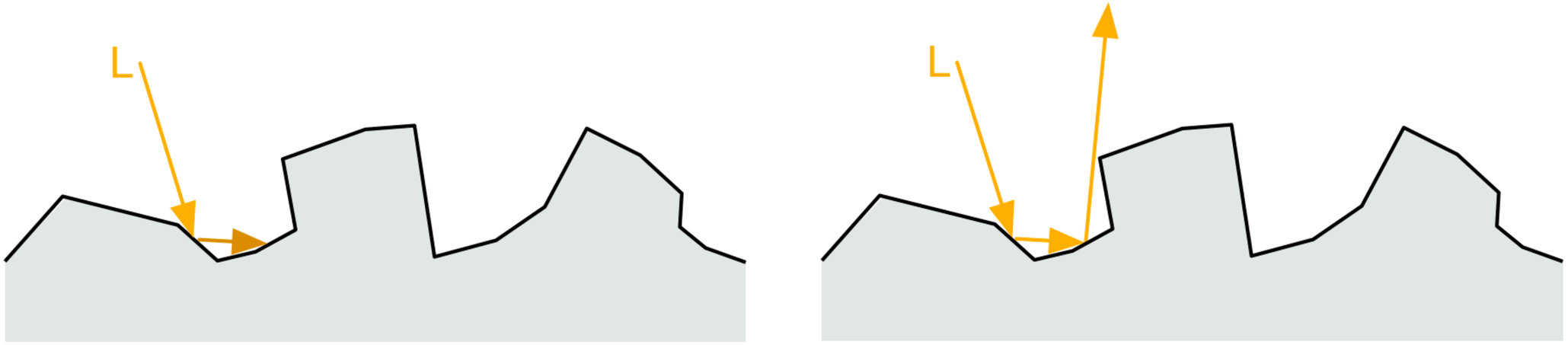
Storing IBLs

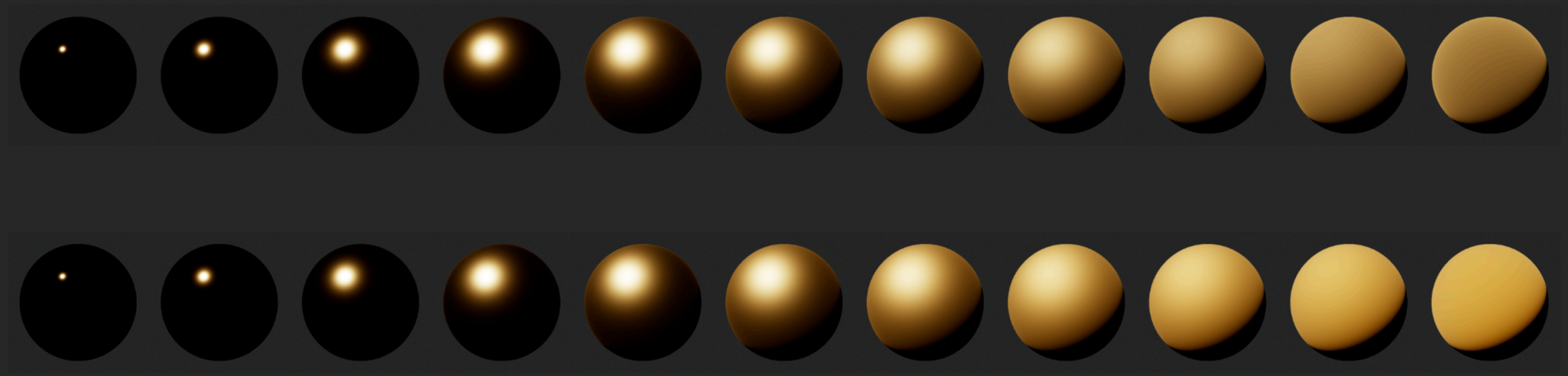
RGBM → quality issues

R11G11B10F → swizzled as RGBA8

Fixing metals




L
L



We use a solution from
Lagarde & Golubev

$$f_r(l,v) = f_{ss}(l,v) + f_{ms}(l,v)$$

$$f_r(l,v) = f_{ss}(l,v) + f_0\left(\frac{1}{r} - 1\right) f_{ss}(l,v)$$

$$r = \int_\Omega D(l,v)V(l,v)\langle n \cdot l \rangle \, dl$$

```
const float V = Visibility(…) * NoL * (VoH / NoH);
const float F = pow5(1.0f - VoH);
r.x += V * (1.0f - F);
r.y += V * F;
```

```
const float V = Visibility(…) * NoL * (VoH / NoH);
const float F = pow5(1.0f - VoH);
r.x += V * F;
r.y += V;
```

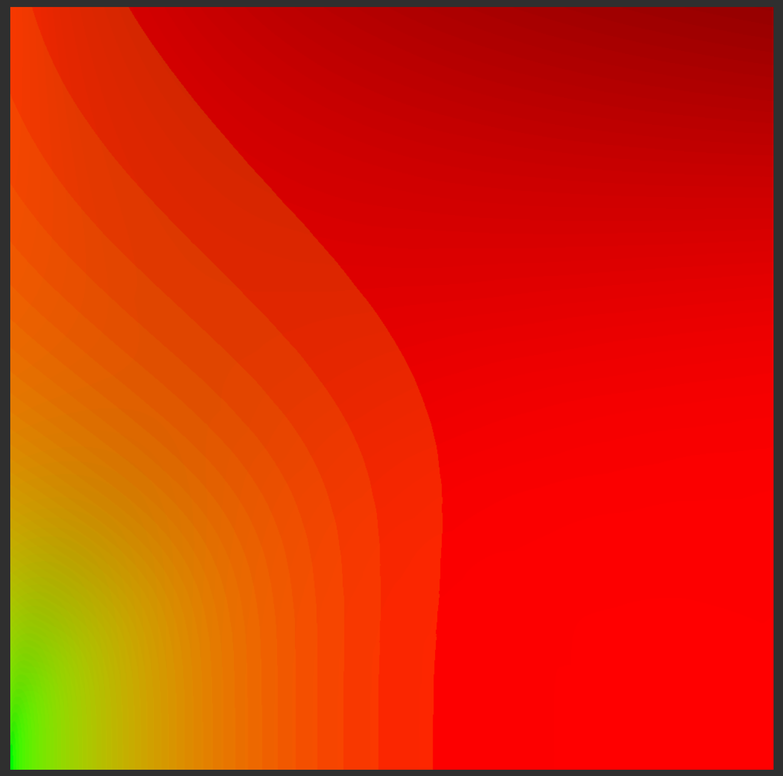
```
vec2 dfg = textureLod(dfgLut, vec2(NoV, roughness), 0.0).xy;
// For image-based lighting
vec3 iblSpecularColor = mix(dfg.xxx, dfg.yyy, f0);

// For other lights
vec3 energyCompensation = 1.0 + f0 * (1.0 / dfg.y - 1.0);
Fr *= pixel.energyCompensation;
```

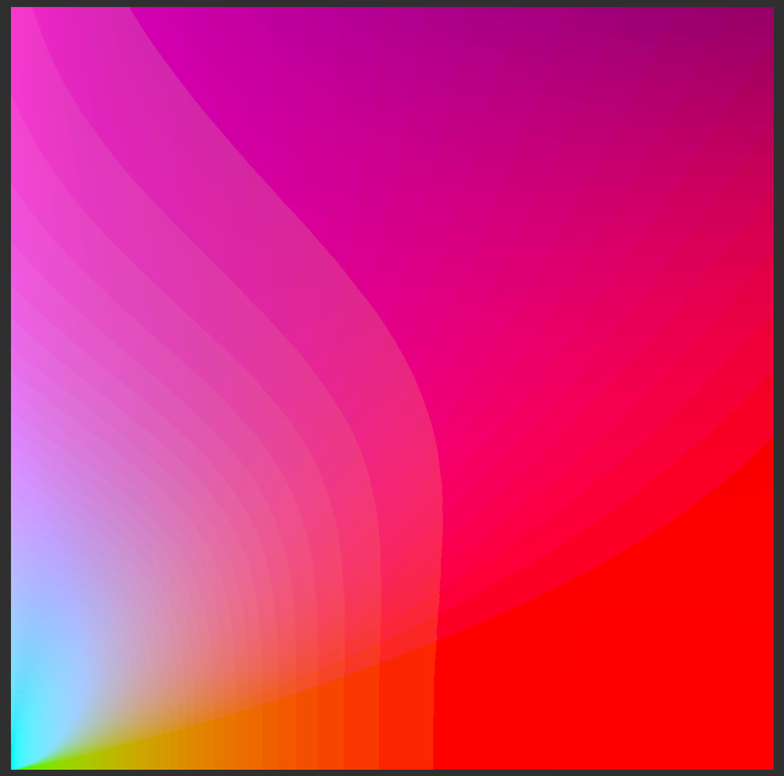
Rendering cloth

$$f_r(v,h,\alpha) = \frac{D_{velvet}(v,h,\alpha)}{4(n\cdot l + n\cdot v - (n\cdot v)(n\cdot l))}$$

$$D_{velvet}(m) = \frac{(2+\frac{1}{\alpha})sin(\theta)^{\frac{1}{\alpha}}}{2\pi}$$



DFG LUT



DFG LUT with cloth BRDF

```
static float DFV_Charlie_Uniform(float NoV, float roughness, size_t numSamples) {
    float r = 0.0f;
    const float3 V(std::sqrt(1.0f - NoV * NoV), 0.0f, NoV);

    for (size_t i = 0; i < numSamples; i++) {
        const float2 u = hammersley(uint32_t(i), 1.0f / numSamples);
        const float3 H = hemisphereUniformSample(u);
        const float3 L = 2 * dot(V, H) * H - V;
        const float VoH = saturate(dot(V, H));
        const float NoL = saturate(L.z);
        const float NoH = saturate(H.z);

        if (NoL > 0.0f) {
            const float V = VisibilityAshikhmin(NoV, NoL, roughness);
            const float D = DistributionCharlie(NoH, roughness);
            r += V * D * NoL * VoH;
        }
    }

    return r * (4.0f * 2.0f * (float) F_PI / numSamples);
}
```

Multi-bounce AO

```
vec3 gtaoMultiBounce(float visibility, const vec3 albedo) {
    // Jimenez et al. 2016,
    // “Practical Realtime Strategies for Accurate Indirect Occlusion"
    vec3 a =  2.0404 * albedo - 0.3324;
    vec3 b = -4.7951 * albedo + 0.6417;
    vec3 c =  2.7552 * albedo + 0.6903;

    return max(
        vec3(visibility), ((visibility * a + b) * visibility + c) * visibility
    );
}

diffuseLobe *= gtaoMultiBounce(ao, diffuseColor);
```

Specular anti-aliasing

```
float normalFiltering(float perceptualRoughness, const vec3 worldNormal) {
    // Kaplanyan 2016, "Stable specular highlights"
    // Tokuyoshi 2017, "Error Reduction and Simplification for Shading Anti-Aliasing"
    // Tokuyoshi and Kaplanyan 2019, "Improved Geometric Specular Antialiasing"
    vec3 du = dFdx(worldNormal);
    vec3 dv = dFdy(worldNormal);

    float variance = specularAntiAliasingVariance * (dot(du, du) + dot(dv, dv));

    float roughness = perceptualRoughnessToRoughness(perceptualRoughness);
    float kernelRoughness = min(2.0 * variance, specularAntiAliasingThreshold);
    float squareRoughness = saturate(roughness * roughness + kernelRoughness);

    return roughnessToPerceptualRoughness(sqrt(squareRoughness));
}

materialRoughness = normalFiltering(
        materialRoughness, getWorldGeometricNormalVector());
```

Ambient occlusion

Scalable Ambient Obscurance

excellent quality → only 7 samples

great performance → 2.3ms @ 1080p on Pixel 4


Le Petit Coin
Restaurant
Le Petit Coin
Restaurant
Le Petit Coin
MENU


Le Petit Coin
Restaurant
Le Petit Coin
Restaurant
Le Petit Coin
MENU

Our changes

interleaved gradient noise → cheaper

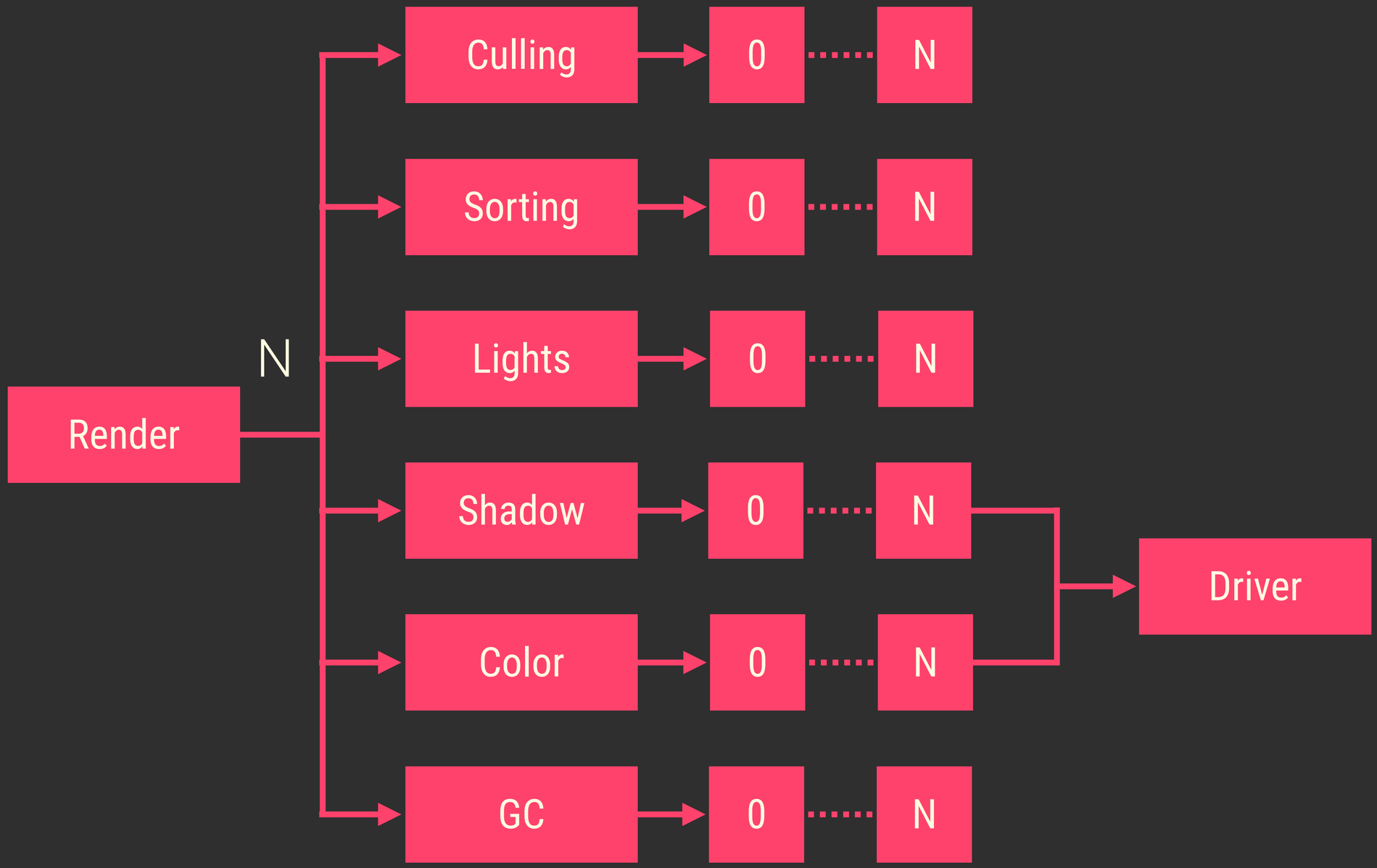
constant spiral angle → avoid sin/cos

face normals from depth → without derivatives

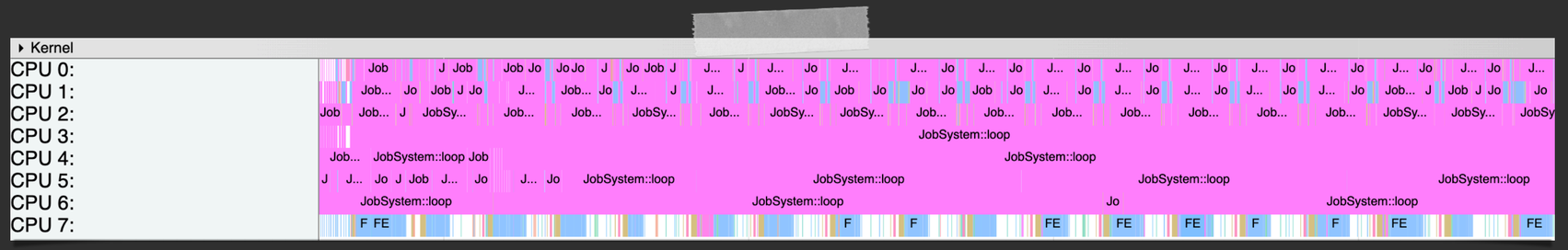
fp16 friendly → cheaper

Job system


Le Petit Coin
MENU


Render
N
Culling
Sorting
Lights
Shadow
Color
GC
0
N
Driver

Render loop	1.15ms
Driver thread	8.5ms


Kernel
CPU 0:
CPU 1:
CPU 2:
CPU 3:
CPU 4:
CPU 5:
CPU 6:
CPU 7:
JobSystem::loop

```
do {
    if (!execute(*state)) {
        std::unique_lock<Mutex> lock(mWaiterLock);
        while (!exitRequested() && !hasActiveJobs()) {
            wait(lock);
            setThreadAffinityById(state->id);
        }
    }
} while (!exitRequested());
```

```
do {
    if (!execute(*state)) {
        std::unique_lock<Mutex> lock(mWaiterLock);
        while (!exitRequested() && !hasActiveJobs()) {
            wait(lock);
            setThreadAffinityById(state->id);
        }
    }
} while (!exitRequested());
```

```
// Highest affinity bit, assuming this is a big core
// This core is not used by the JobSystem
uint32_t id = std::thread::hardware_concurrency() - 1;

while (true) {
    JobSystem::setThreadAffinityById(id);
    if (!execute()) {
        break;
    }
}
```

```
void JobSystem::setThreadAffinityById(size_t id) noexcept {
#if defined(__linux__)
    cpu_set_t set;
    CPU_ZERO(&set);
    CPU_SET(id, &set);
    sched_setaffinity(gettid(), sizeof(set), &set);
#endif
}
```

@romainguy

Physically based rendering in Filament, Guy & Agopian
https://google.github.io/filament/Filament.html

Scalable Ambient Obscurance, McGuire, 2012
https://casual-effects.com/research/McGuire2012SAO/index.html

PBR Diffuse Lighting for GGX, Hammon, 2017
https://www.gdcvault.com/play/1024478/PBR-Diffuse-Lighting-for-GGX

The road toward unified rendering with Unity's high definition rendering pipeline, Lagarde & Golubev, 2018
http://advances.realtimerendering.com/s2018/index.htm

Crafting a Next-Gen Material Pipeline for The Order: 1886, Neubelt & Pettineo, 2014
https://www.gdcvault.com/play/1020162/Crafting-a-Next-Gen-Material

Production Friendly Microfacet Sheen BRDF, Estevez & Kulla, 2017
http://www.aconty.com/pdf/s2017_pbs_imageworks_sheen.pdf

Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs, Heitz, 2014
http://jcgt.org/published/0003/02/03/paper.pdf

Practical Real-Time Strategies for Accurate Indirect Occlusion, Jimenez, 2016
https://blog.selfshadow.com/publications/s2016-shading-course/

Stable Specular Highlights, Kaplanyan, 2016
http://developer.download.nvidia.com/gameworks/events/GDC2016/akaplanyan_specular_aa.pdf

Error Reduction and Simplification for Shading Anti-Aliasing, Tokuyoshi, 2017
http://www.jp.square-enix.com/tech/library/pdf/Error%20Reduction%20and%20Simplification%20for%20Shading%20Anti-Aliasing.pdf

Improved Geometric Specular Antialiasing, Kaplanyan & Tokuyoshi, 2019
http://www.jp.square-enix.com/tech/library/pdf/ImprovedGeometricSpecularAA.pdf

HDRIs/environments

https://hdrihaven.com/

Textures

https://texturehaven.com/ & https://www.cgbookcase.com/

glTF Samples

https://github.com/KhronosGroup/glTF-Sample-Models

Amazon Lumberyard Bistro

https://developer.nvidia.com/orca/amazon-lumberyard-bistro

Lee Perry-Smith Head Scan
http://graphics.cs.williams.edu/data/

Statue (“Lucy”) by Stanford Computer Graphics Laboratory
http://www.graphics.stanford.edu/data/3Dscanrep/#uses