
THE ORDER
1886
Readyatdawn
STUDIOS

Crafting a Next-Gen Material Pipeline for The Order: 1886

David Neubelt, Ready at Dawn Studios
Matt Pettineo, Ready at Dawn Studios

Readyatdawn STUDIOS

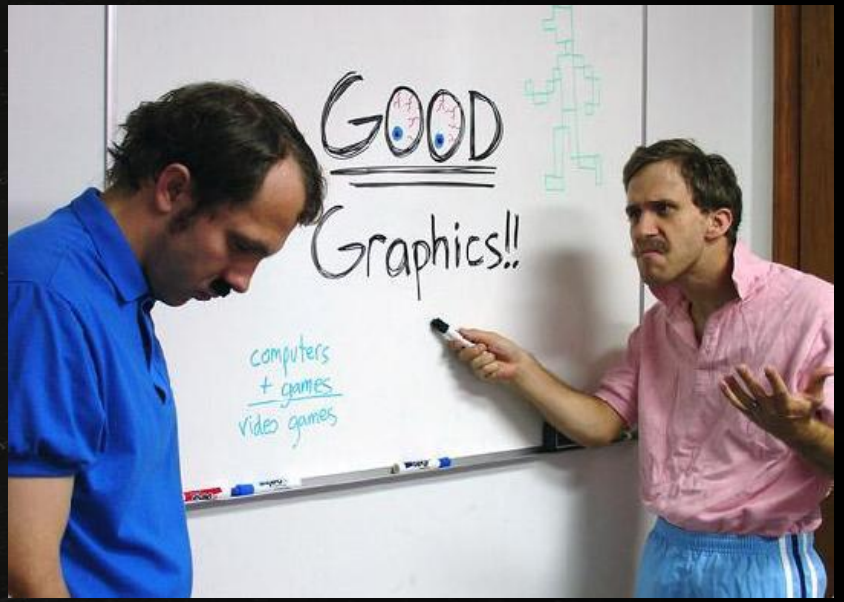
About Our Project

- Brand-new IP
- Alternate history 19th-century London
- PS4 Exclusive
- ~100 developers

Readyatdawn STUDIOS

About Our Project

- Work began in early 2011
- Originally just 2 graphics programmers
  - Now we have 5!



Readyatdawn STUDIOS

Engine Overview

- Latest version of our in-house engine
- Windows(DX11) and PS4, but heavily geared towards PS4
  - Fine-grained task scheduling
  - Low-overhead, multithreaded cmd buffer generation
  - PS4 Secret Sauce™
- Physically based rendering from the start

Engine Overview

- Core tools framework uses C++ and Qt
  - Built on common UI core
  - Can be embedded in Maya
- Custom build pipeline built in C++
  - Major engineering effort
  - Heavily multithreaded
  - Processed assets cached on local servers

Engine Overview

- Renderer embedded in Maya
  - Uses DX11 Viewport 2.0 override (Maya 2014)
  - Render using their device instead of our own
  - Tell Maya which UI to draw
- Materials/particles/lens flares/etc. can be live edited inside of Maya
- Serves as primary level editor

Memory

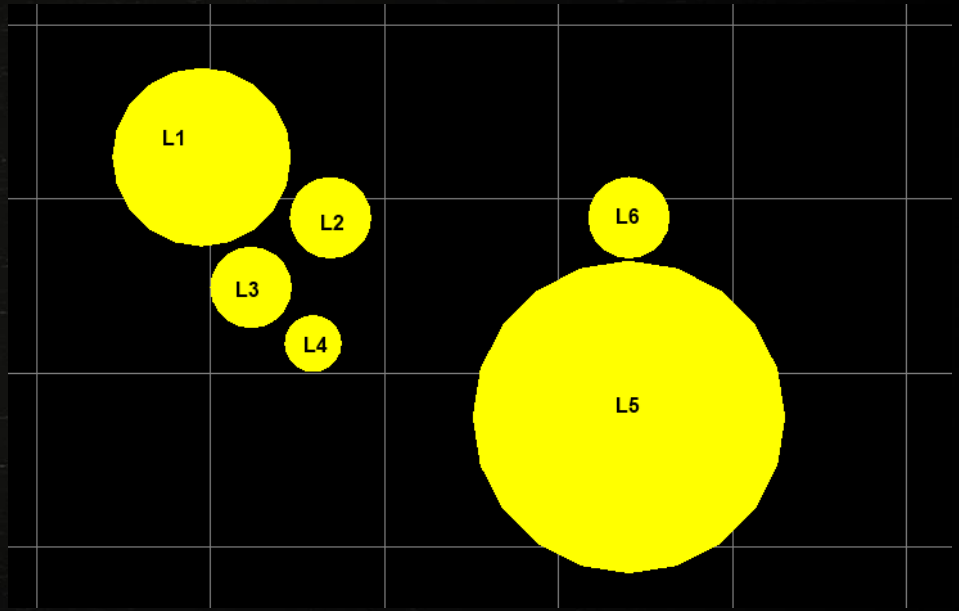
- 2 GB texture budget
- 128 MiB sound budget
- 700 MiB level geometry
- 600 MiB character textures
- 250 MiB global textures (FX, UI, light maps, etc..)
  - Even characters have lightmap data
- 700 MiB animation

Memory

- Environmental art is mapped at 512 pixels per unit
- Standard environment tilling texture is 1024x1024

Lighting Pipeline

- Tiled forward lighting (AKA Forward+)
  - Depth prepass
  - CS bins lights per tile
  - Two depth partitions
  - Punctual light sources (no area lights…yet!)



Lighting Pipeline

- Transparents fully lit with diffuse + specular
  - Separate depth prepass + depth buffer for transparents
  - Transparent light list generated per-tile
    - TileMinDepth = TileMin(transparentDepth)
    - TileMaxDepth = TileMax(opaqueDepth)
  - Also support per-vertex lighting for particles

Lighting Pipeline

- Lightmaps for static geo
  - Baked on GPU farm using Optix
  - H-basis for directional variation
- SH probes for dynamics
  - 3rd-order (9 coefficients)
- Pre-convolved specular probes
  - Cubemaps rendered in-engine, convolved in a compute shader

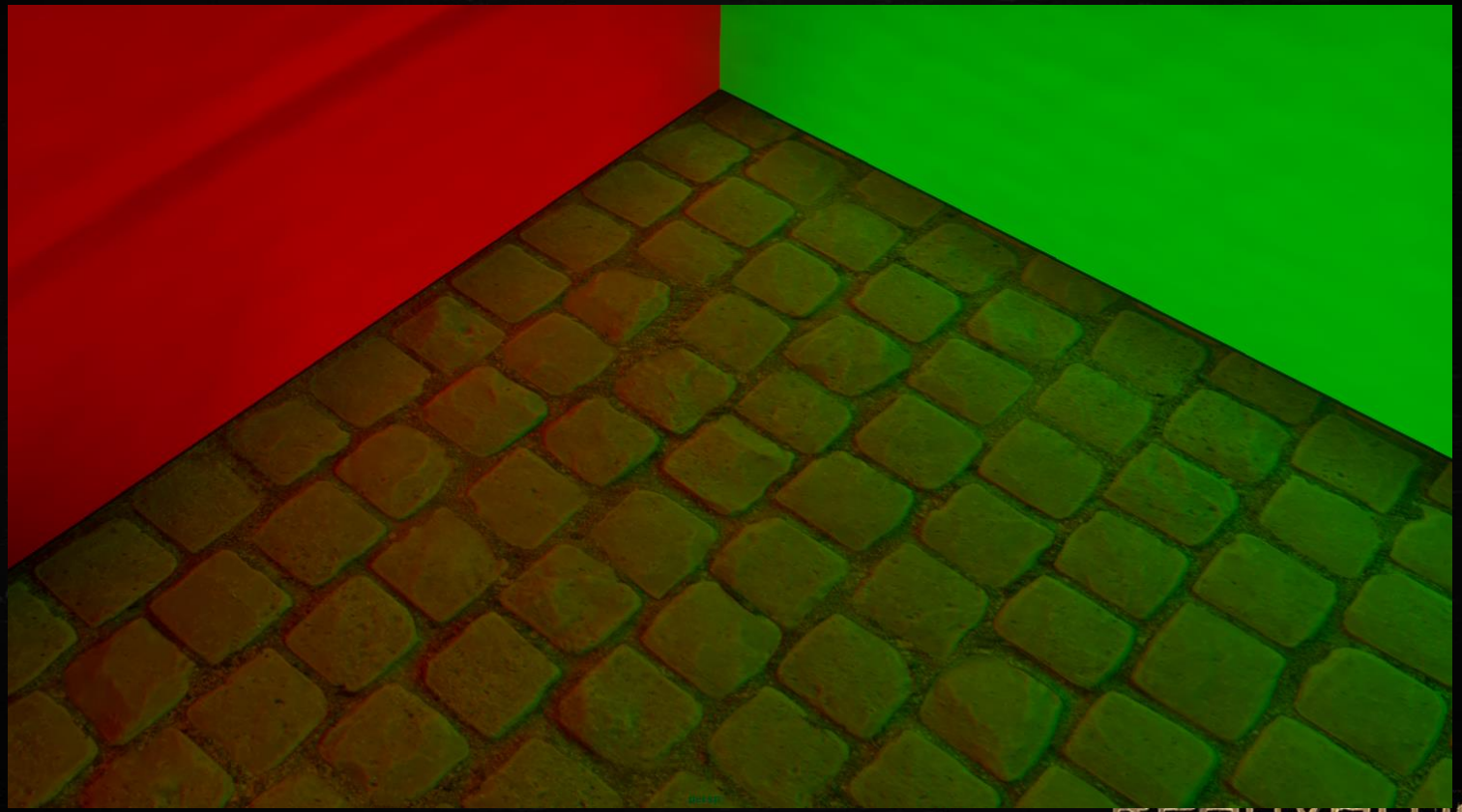
Lighting Pipeline

Lighting Pipeline

Lighting Pipeline

Lighting Pipeline

Lighting Pipeline



Ambient Occlusion

- Directional AO maps (H-basis) baked for characters and static geo
- Static geo only uses it to occlude specular from probes
- Dynamic geo applies AO to diffuse from SH probes

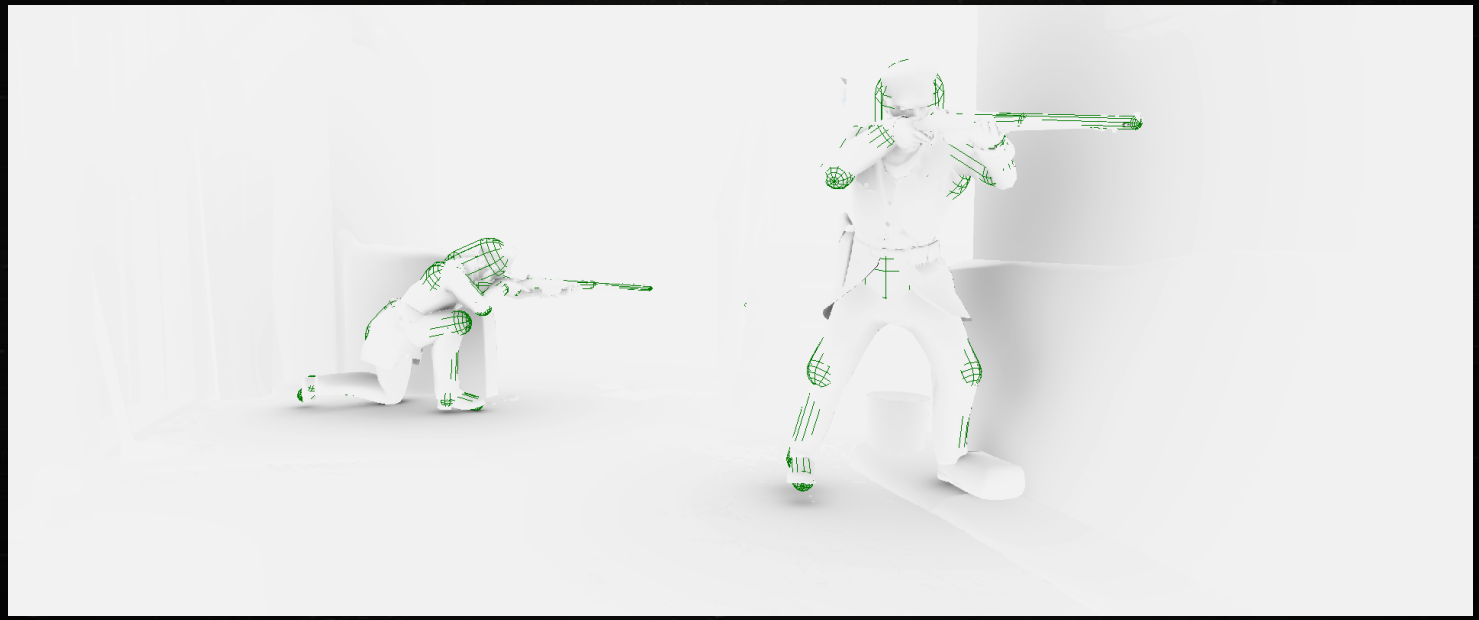
Without Reflection Occlusion

With Reflection Occlusion

Reflection Occlusion Visualization

Ambient Occlusion

- Dynamic AO from capsules skinned to characters



AO Capsules Off

Readyatdawn STUDIOS

AO Capsules On

Readyatdawn STUDIOS

Core Shading Model

- Default specular BRDF is Cook-Torrance
  - D term is GGX distribution from Walter et al.
  - Matching Smith G term derived in same paper
  - Schlick's approximation for Fresnel
  - Lambertian diffuse, balanced with specular intensity

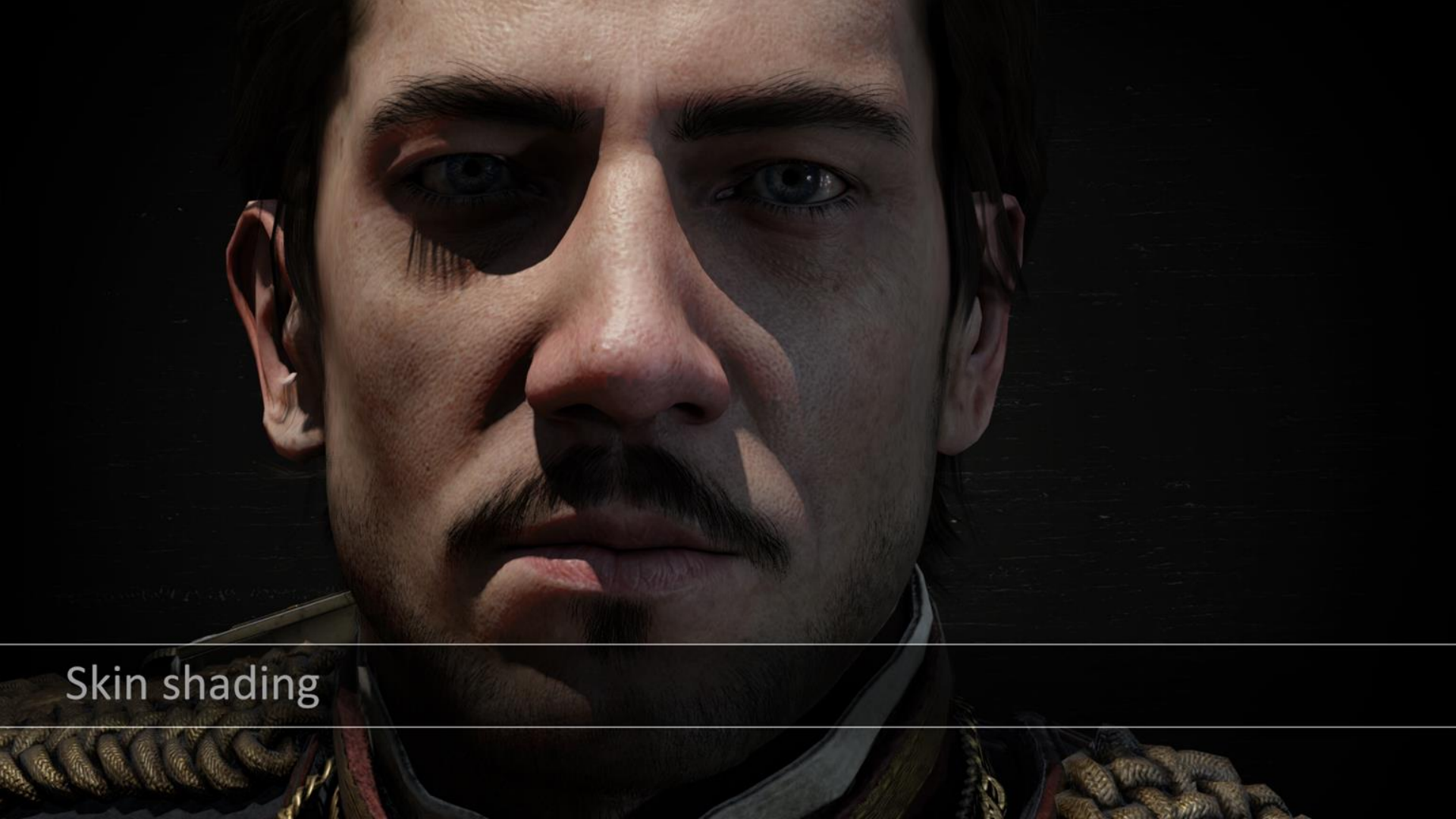
Core Shading Model

- GGX + Cook Torrance == Lots of math
  - But we have lots of ALU!
- Can be optimized
  - Don’t use trig
  - Fold Smith G term into denominator
  - See our course notes from SIGGRAPH
- Have artists work with sqrt(roughness)
  - More intuitive, better for blending

```
// Helper for computing the GGX visibility term
float GGX_V1(in float m2, in float nDotX) {
    return 1.0f / (nDotX + sqrt(m2 + (1 - m2) * nDotX * nDotX));
}

// Computes the specular term using a GGX microfacet distribution. m is roughness, n is the surface normal, h is the
// half vector, and l is the direction to the light source
float GGX_Specular(in float m, in float3 n, in float3 h, in float3 v, in float3 l) {
    float nDotH = saturate(dot(n, h));
    float nDotL = saturate(dot(n, l));
    float nDotV = saturate(dot(n, v));

    float nDotH2 = nDotH * nDotH;
    float m2 = m * m;

    // Calculate the distribution term
    float d = m2 / (Pi * pow(nDotH * nDotH * (m2 - 1) + 1, 2.0f));

    // Calculate the matching visibility term
    float v1i = GGX_V1(m2, nDotL);
    float v1o = GGX_V1(m2, nDotV);
    float vis = v1i * v1o;

    // Multiply this result with the Fresnel term
    return d * vis;
}
```

Core Shading Model

- Other BRDFs available
  - Beckmann (also taken from Walter et al.)
  - Anisotropic GGX
  - Hair (Kajiya-Kay)
  - Skin (pre-integrated diffuse)
  - Cloth



Skin shading

Skin

- Our most expensive shader!
  - Texture lookup per light based on N dot L
  - Multiple specular lobes
- Didn’t use shader gradient-based curvature
  - Too many artifacts
  - Used curvature maps instead

Pre-integrated skin diffuse with SH

$$D(\theta, r) = \frac{\int_{-\pi}^{\pi} \cos(\theta + x) R(2r \sin(x/2)) dx}{\int_{-\pi}^{\pi} R(2r \sin(x/2)) dx}$$

$$D_n(r) = 2\pi \int_0^{\pi/2} Y_{n,0}(\theta) D(\theta, r) sin(\theta) d\theta$$

$$f_{skinSH}(\mathbf{n}, r) = \sum_{l=0}^{2} \sum_{m=-1}^{l} \sqrt{\frac{4\pi}{2l+1}} D_l(r) L_{l,m} Y_{l,m}(n)$$

Ambient Skin (Diffuse only)

Left is using normal SH lighting convolved with cosine kernel

Right is using SH lighting convolved with the scattering kernel

Hair Shading

- Not physically-based ☹
- Kajiya-Kay
  - Haven't had time to investigate real-time Marschner
- Tweaked Fresnel curve
- SH diffuse using tangent direction
  - Analytic Tangent Irradiance Environment Maps for Anisotropic Surfaces[Mehta 2012]

Hair Shading

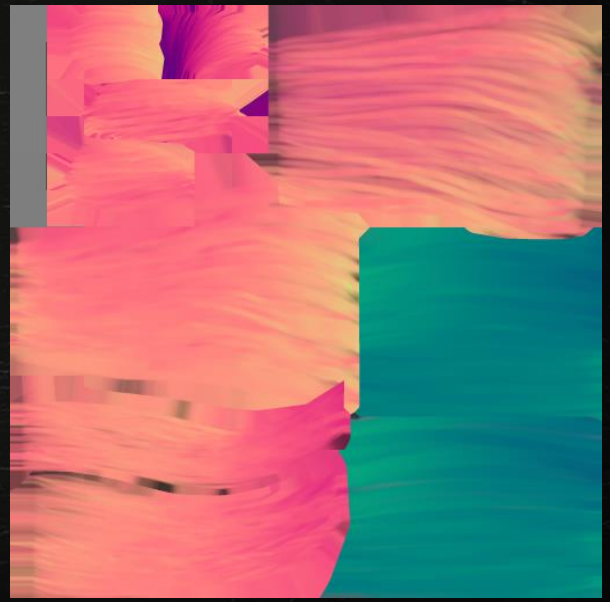
- Secondary specular lobe
  - Shifted along tangent towards tips
  - Takes albedo color

Hair Shading

- Shift maps to break up highlights
  - Additional shift along tangent direction

Hair Shading

- Flow maps to define tangent direction
  - Authored in Mari

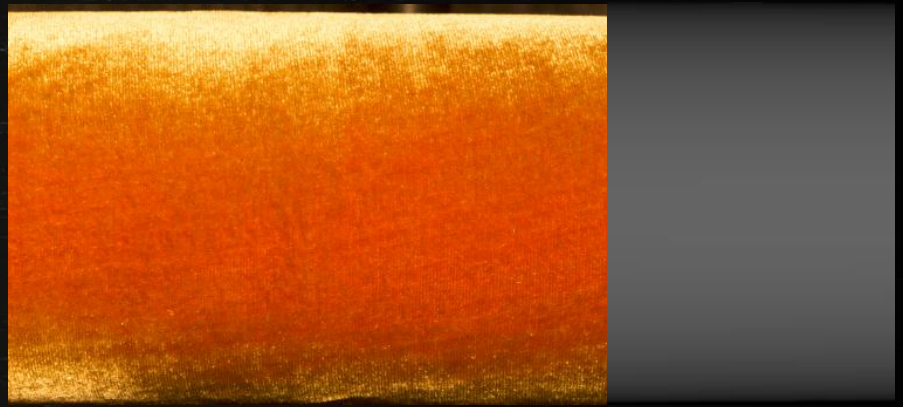


Cloth Shading

Cloth Shading

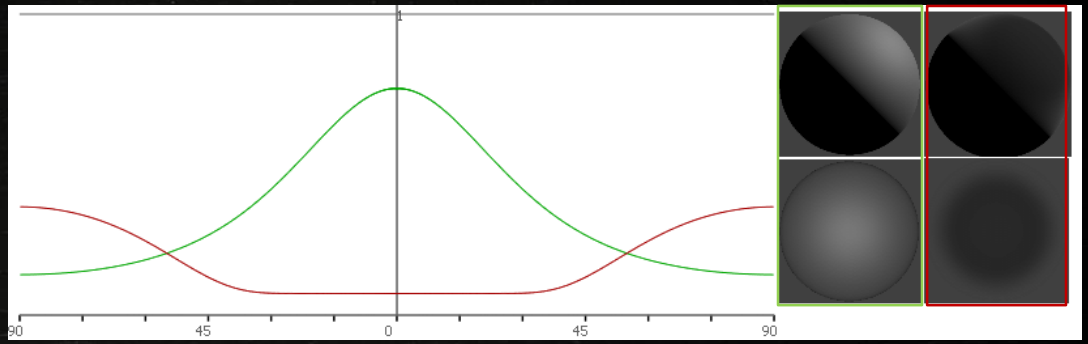
Observations from photo reference:

- Soft specular lobe with large smooth falloffs
- Fuzz on the rim from asperity scattering
- Low specular contribution at front facing angles
- Some fabrics have two toned specular colors

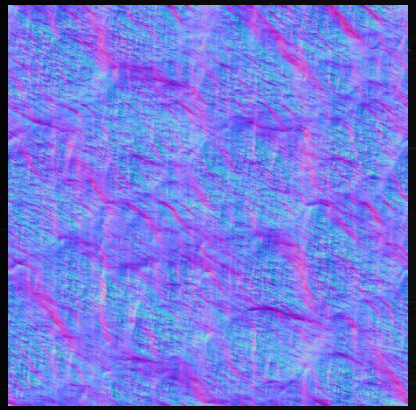


Cloth Shading

- Inverted Gaussian for asperity scattering
- Translation from origin to give more specular at front facing angles
- No geometry term



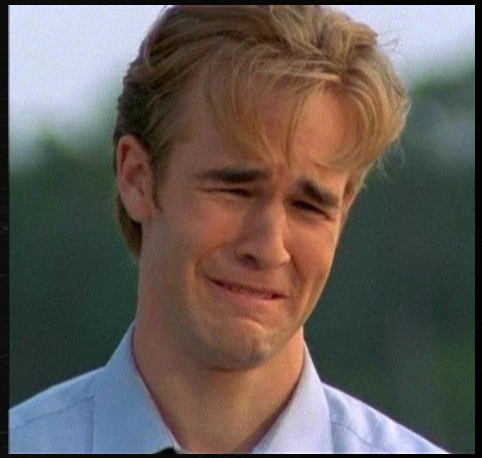
Specular Aliasing



+

$$f(\mathbf{l},\mathbf{v}) = \frac{F(\mathbf{v},\mathbf{h})\,G(\mathbf{l},\mathbf{v},\mathbf{h})\,D(\mathbf{h})}{4\,(\mathbf{n}\cdot\mathbf{l})\,(\mathbf{n}\cdot\mathbf{v})}$$

=



Specular Aliasing

- Modify roughness maps to reduce aliasing
- Using technique based on “Frequency Domain Normal Map Filtering” by Han et al.

$$f^{\text{eff}}(\mathbf{l},\mathbf{v};\gamma) = \int_{\Omega} f(\mathbf{l},\mathbf{v})\gamma(\mathbf{n})d\mathbf{n}$$

$$f_{lm}^{\text{eff}} = \sqrt{\frac{4\pi}{2l+1}} f_l \gamma_{lm}$$

Specular Aliasing

- Represent NDF as spherical Gaussian (vMF distribution)
- Approximate BRDF in SH as a Gaussian
- Convolution of two Gaussians is a new Gaussian
- Use relationship to compute new roughness

BRDF NDF

$$\Lambda_l f_l^{\text{eff}} = e^{(\alpha l)^2} e^{-\frac{l^2}{2\kappa}} = e^{(\alpha' l)^2}$$

New Roughness

$$\alpha' = \sqrt{\alpha^2 + (2\kappa)^{-1}}$$

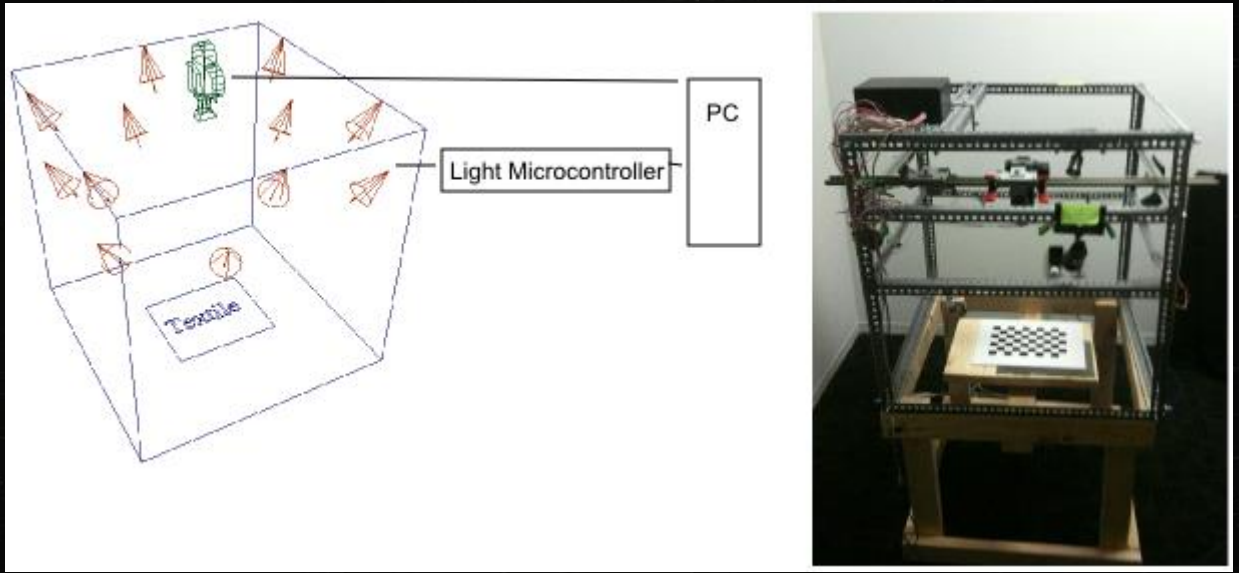
No AA

Ours

Ground Truth

3D Material Scanning

Scanning

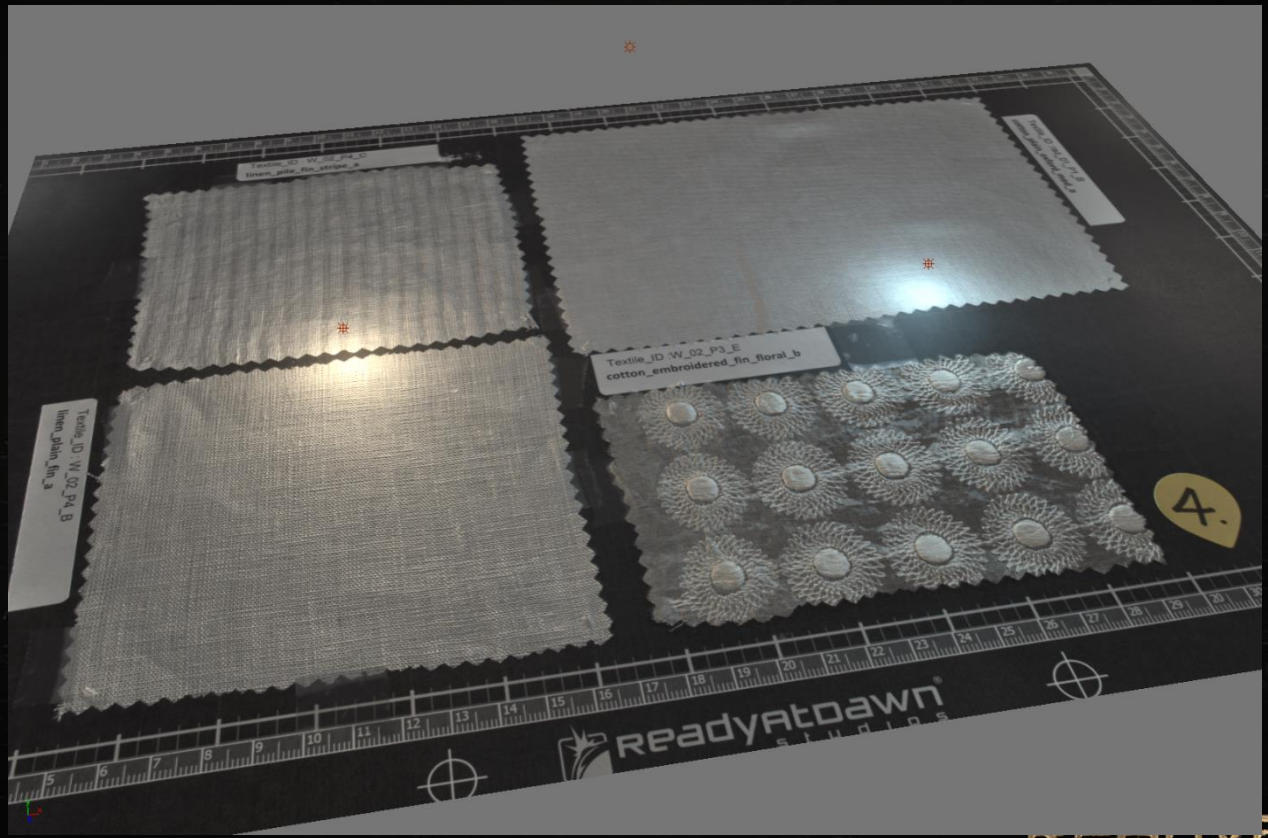


Readyatdawn STUDIOS

Scanning

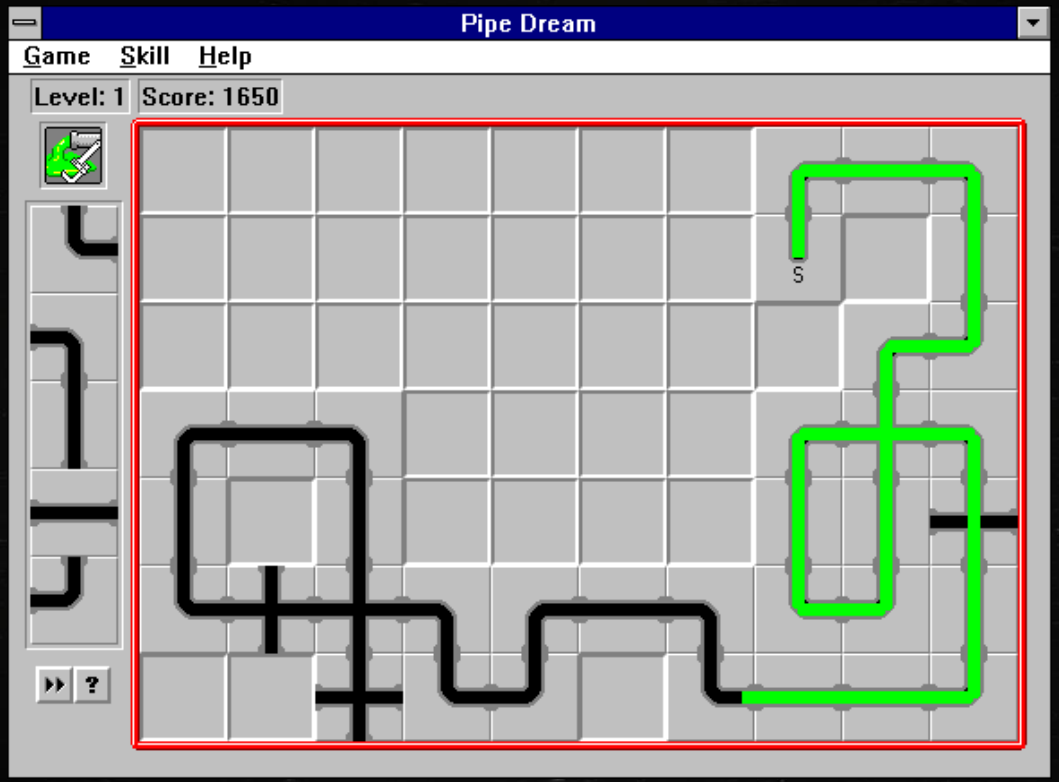
Readyatdawn STUDIOS

Scanning



readyatdawn
STUDIOS

Material Pipeline



Material Pipeline

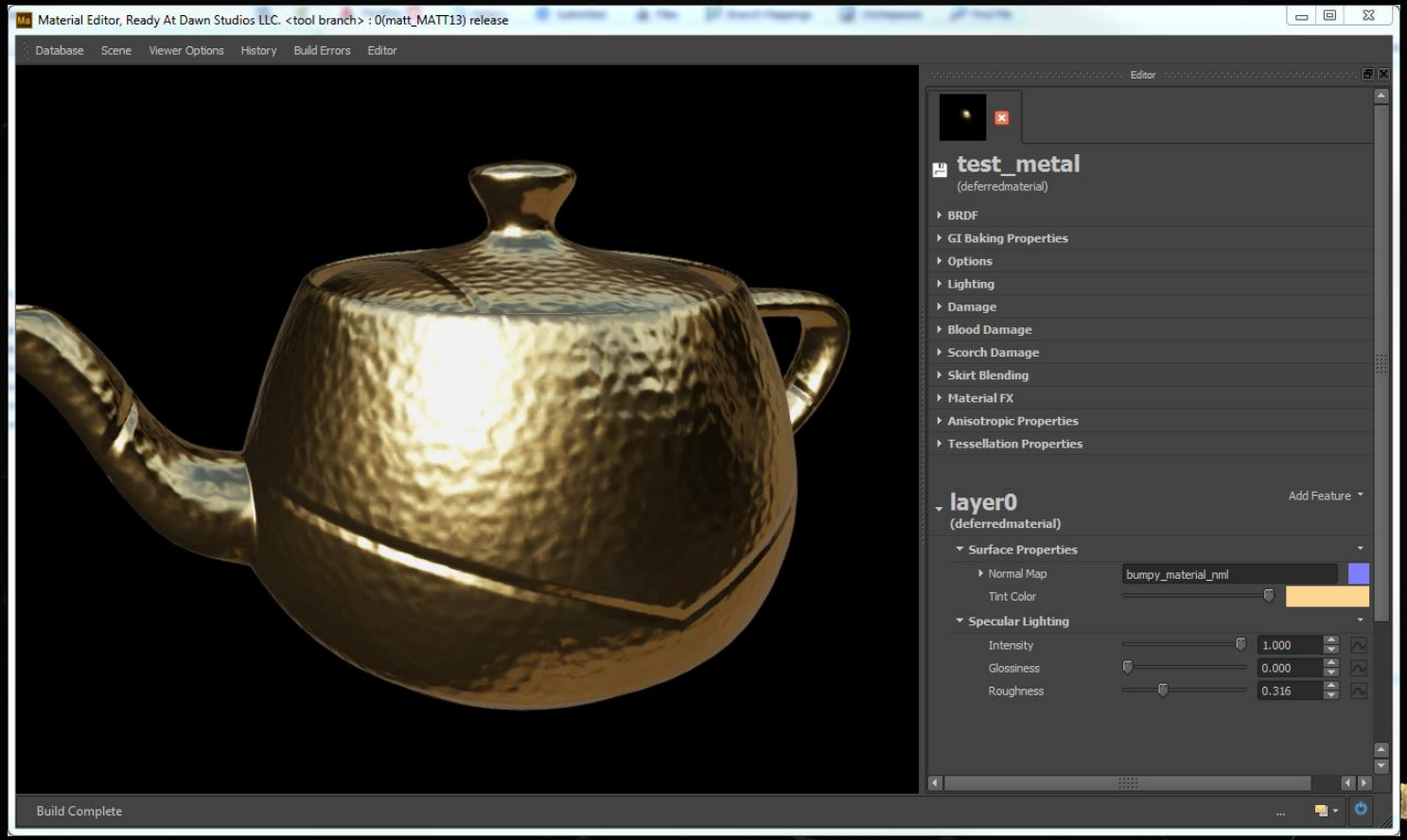
- Materials = text assets
- Uses our custom data language (radattr)
- Language supports:
  - Types
  - Inheritance
  - UI layouts
  - Metadata

```
:bumpy_material := ( opaquematerial
    :enable_diffuse_ = true
    :enable_specular_ = true
    :specular_map = ( :name = "bumpy_material_spc" )
    :normal_map = ( :name = "bumpy_material_nml" )
    :specular_intensity = 0.05
    :specular_roughness = 0.1
)

:bumpy_material_red := ( bumpymaterial
    :albedo_tint_color = ( Color
        :r = 1.0
        :g = 0.0
        :b = 0.0
    )
)
```

Material Pipeline

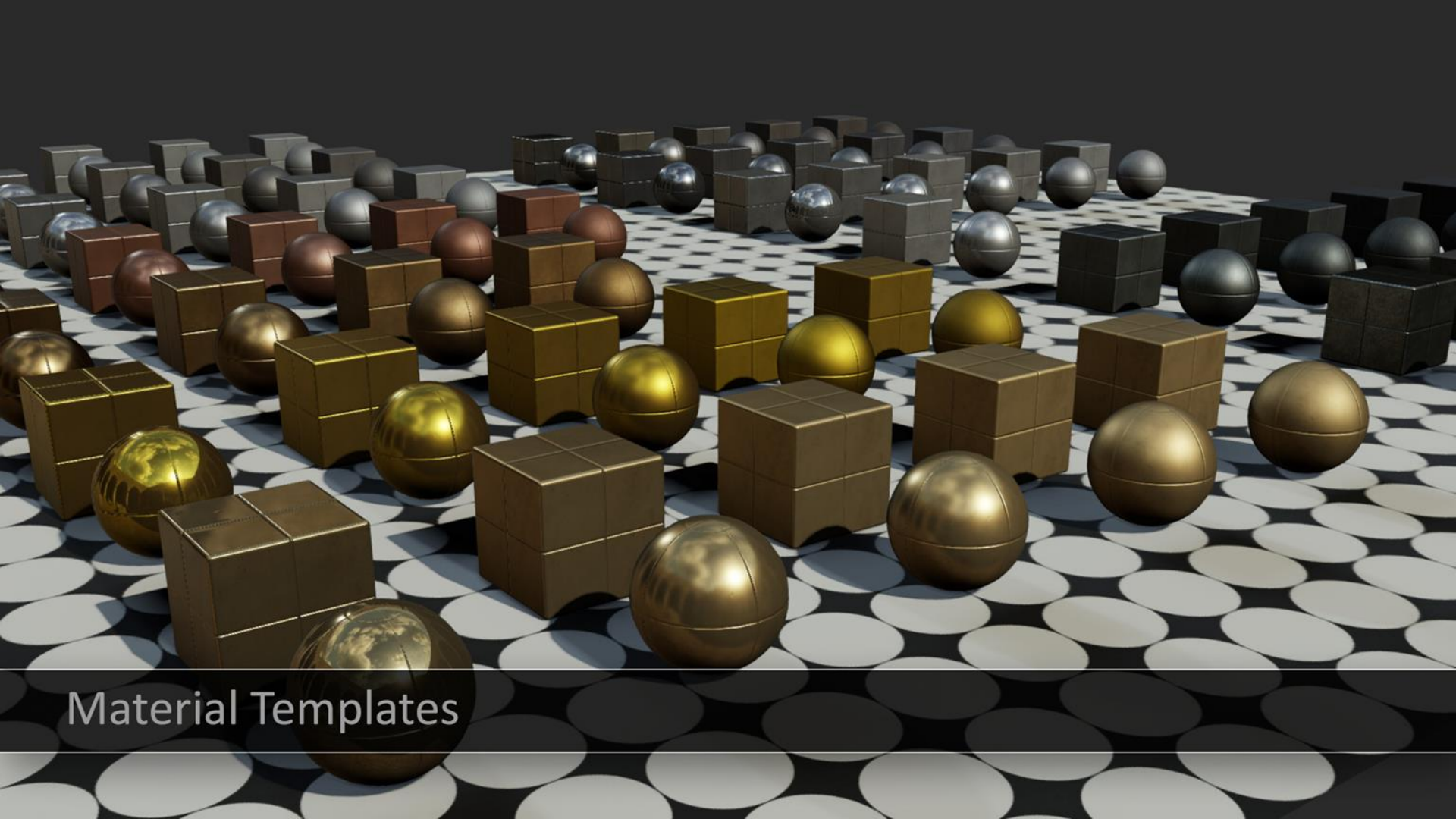
- Material authoring is feature-based
  - No shader trees
- Material editor tool for real-time editing
  - Can also be hosted inside of Maya

Material Pipeline

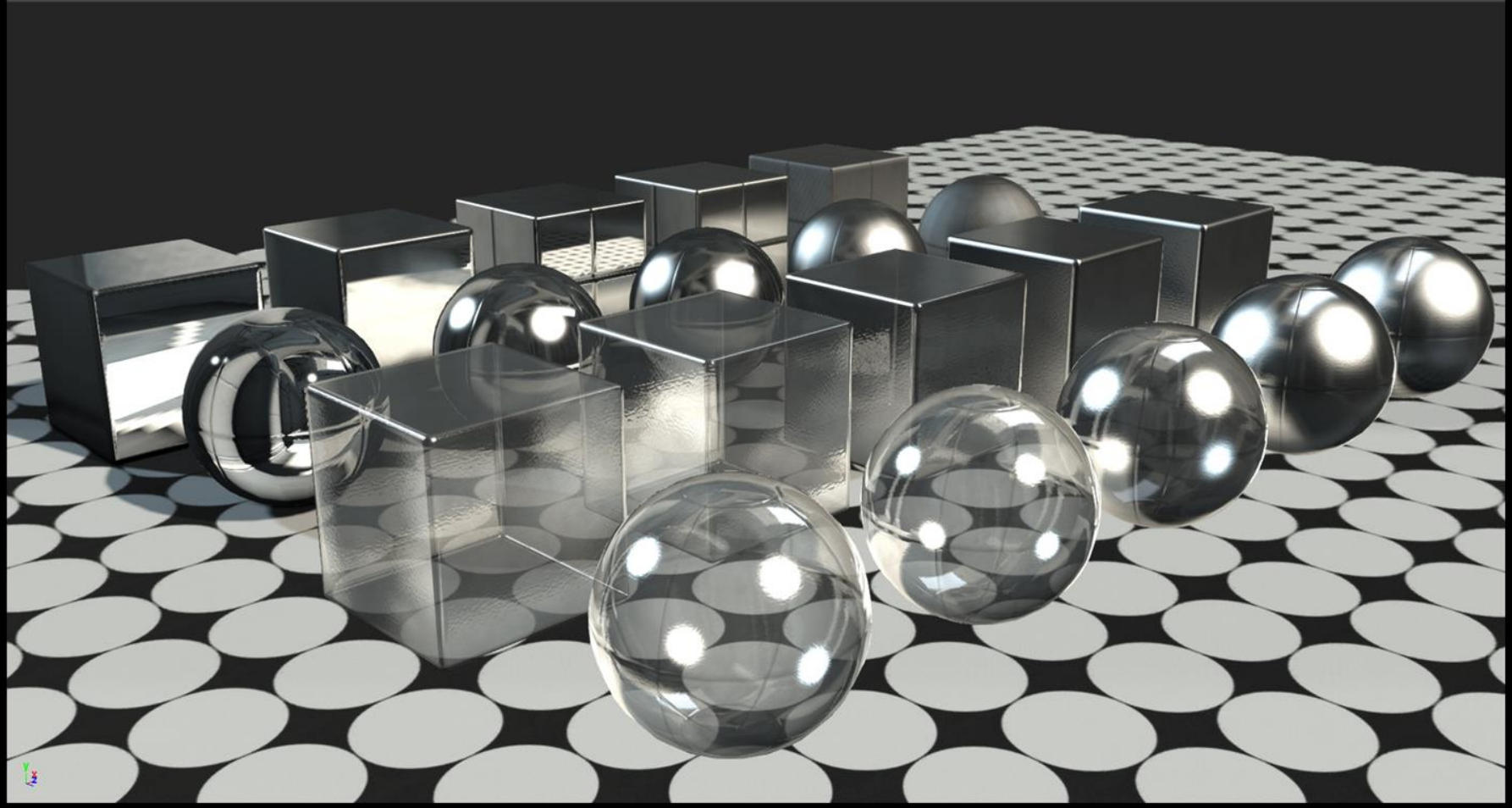


Material Pipeline

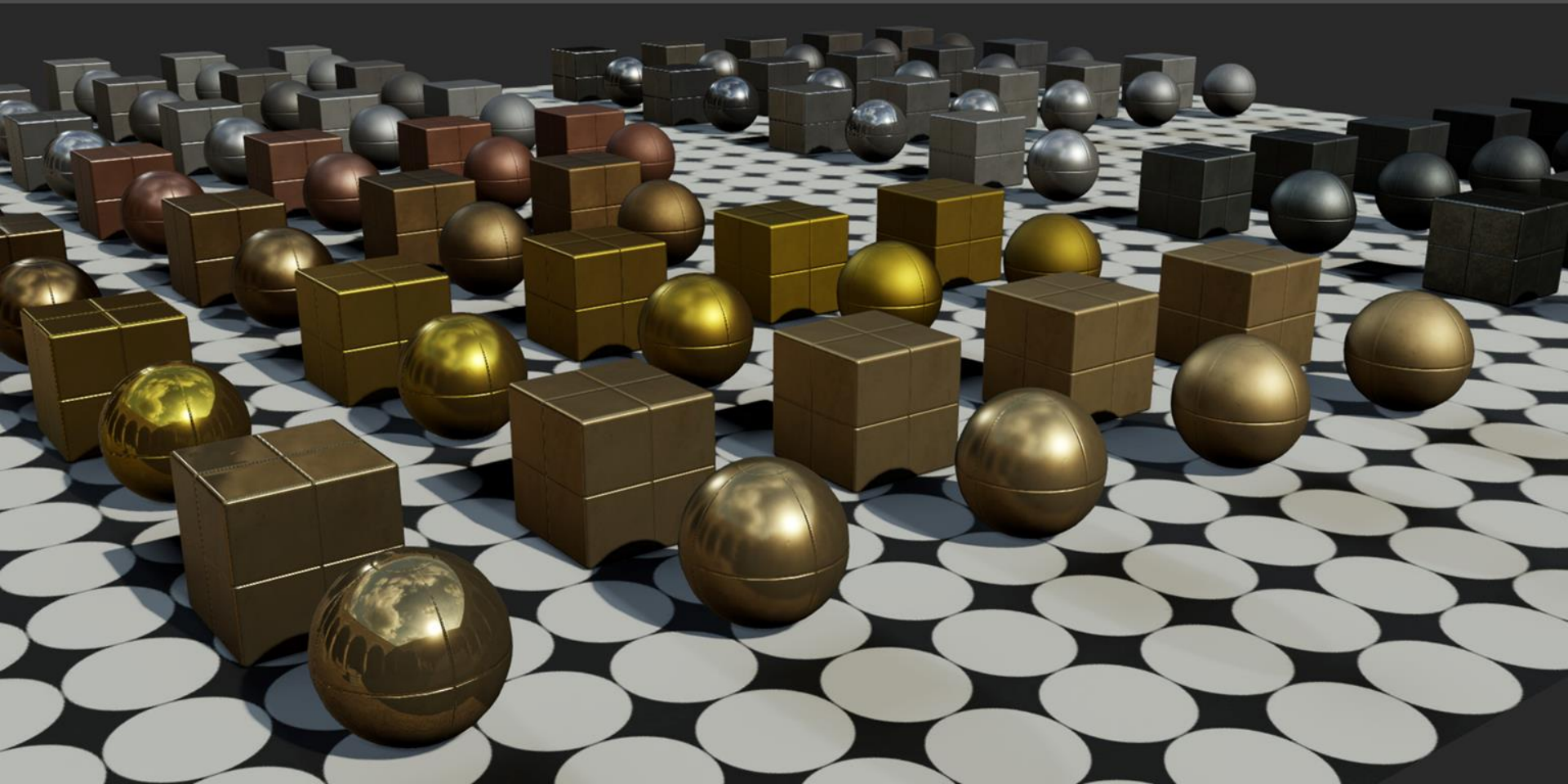
- radattr inheritance used to make *templates*
- Common parameters shared in base material
- Derived material only stores changes from base
- Quicker asset creation
- Global changes can be made in a single asset


Material Templates

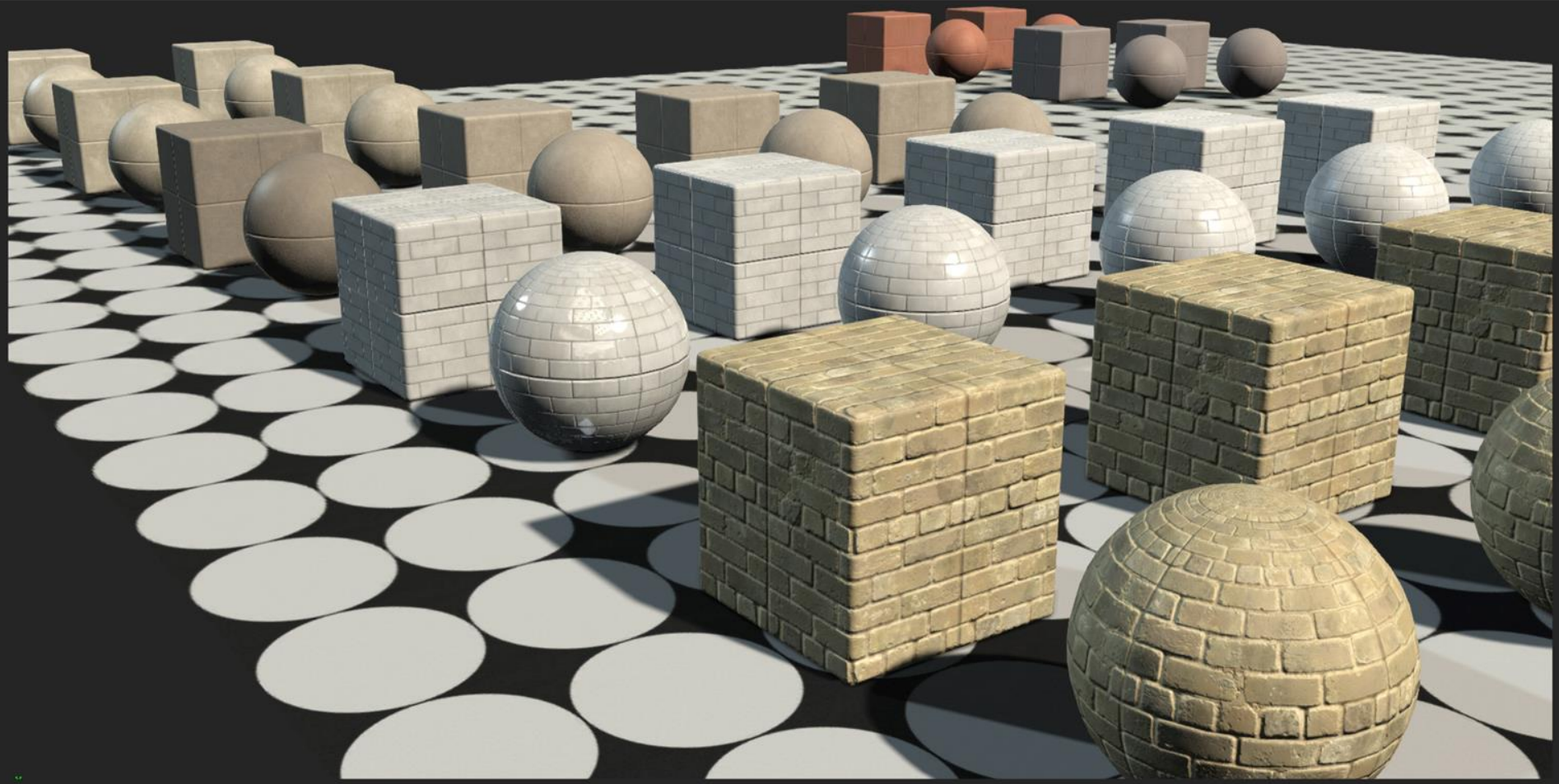
Global Material Templates: Sample Glass Subset



Global Material Templates: Sample Metal Subset

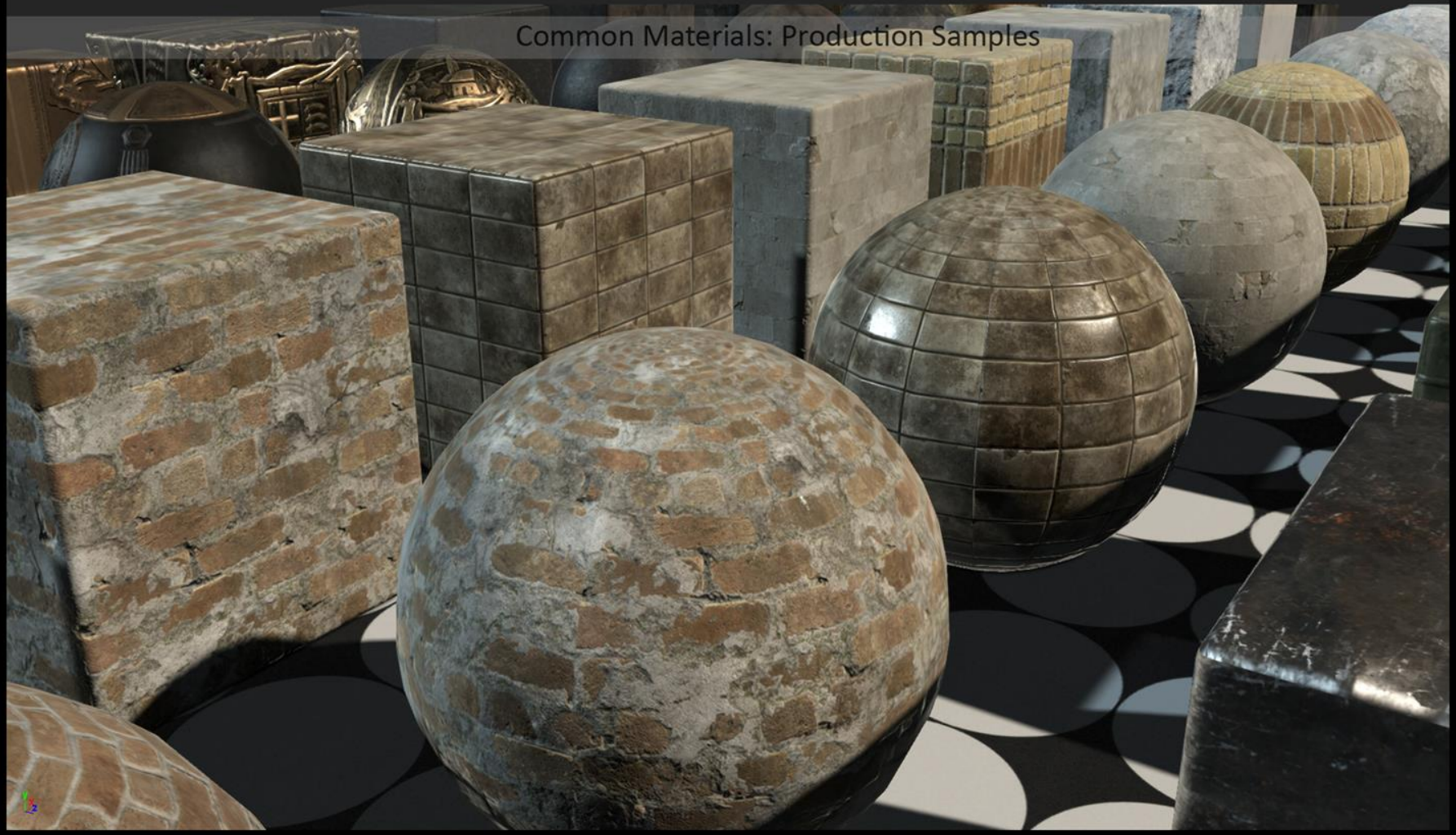


Global Material Templates: Sample Masonry Subset

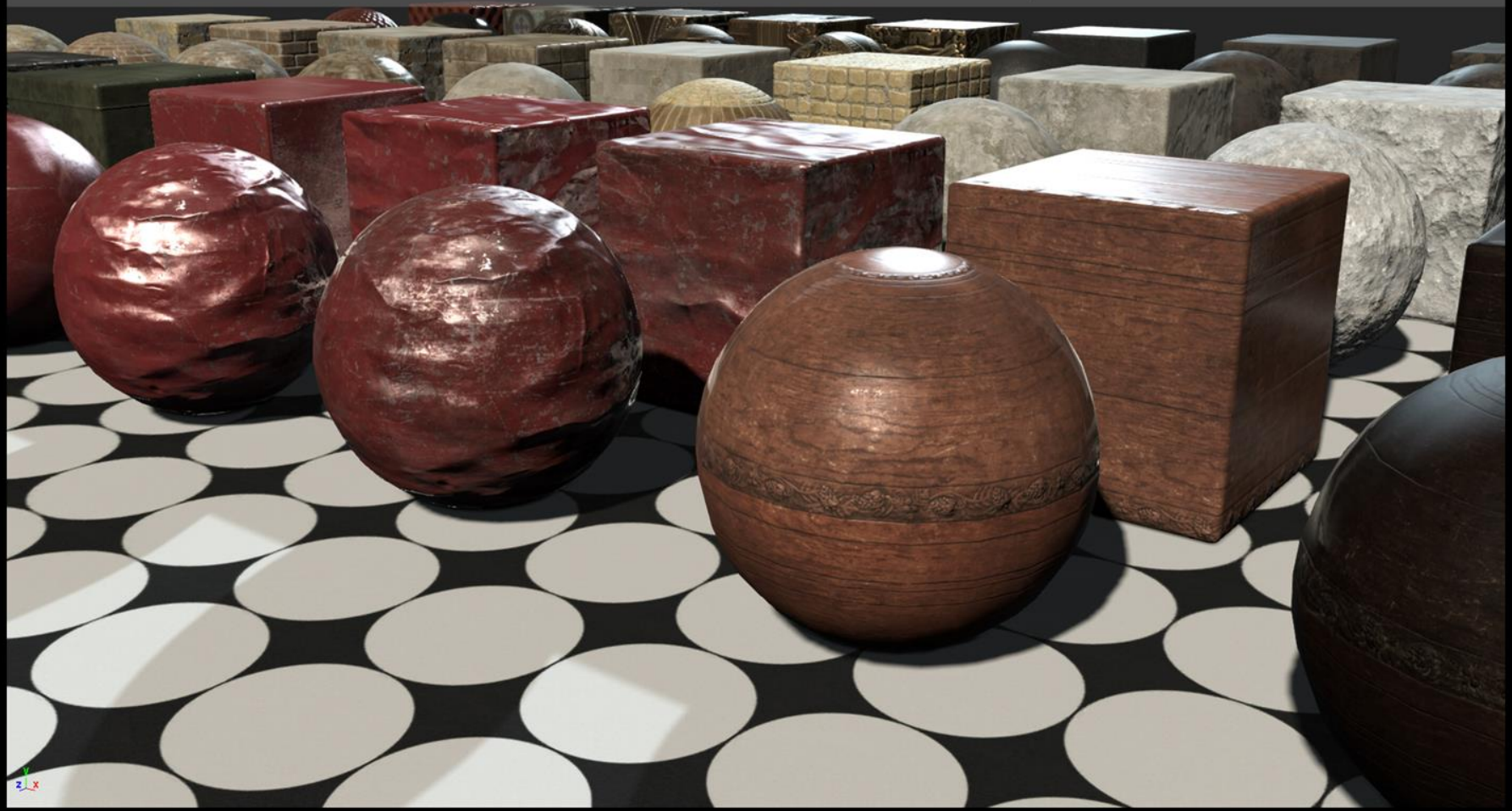


Common Materials: Production Samples

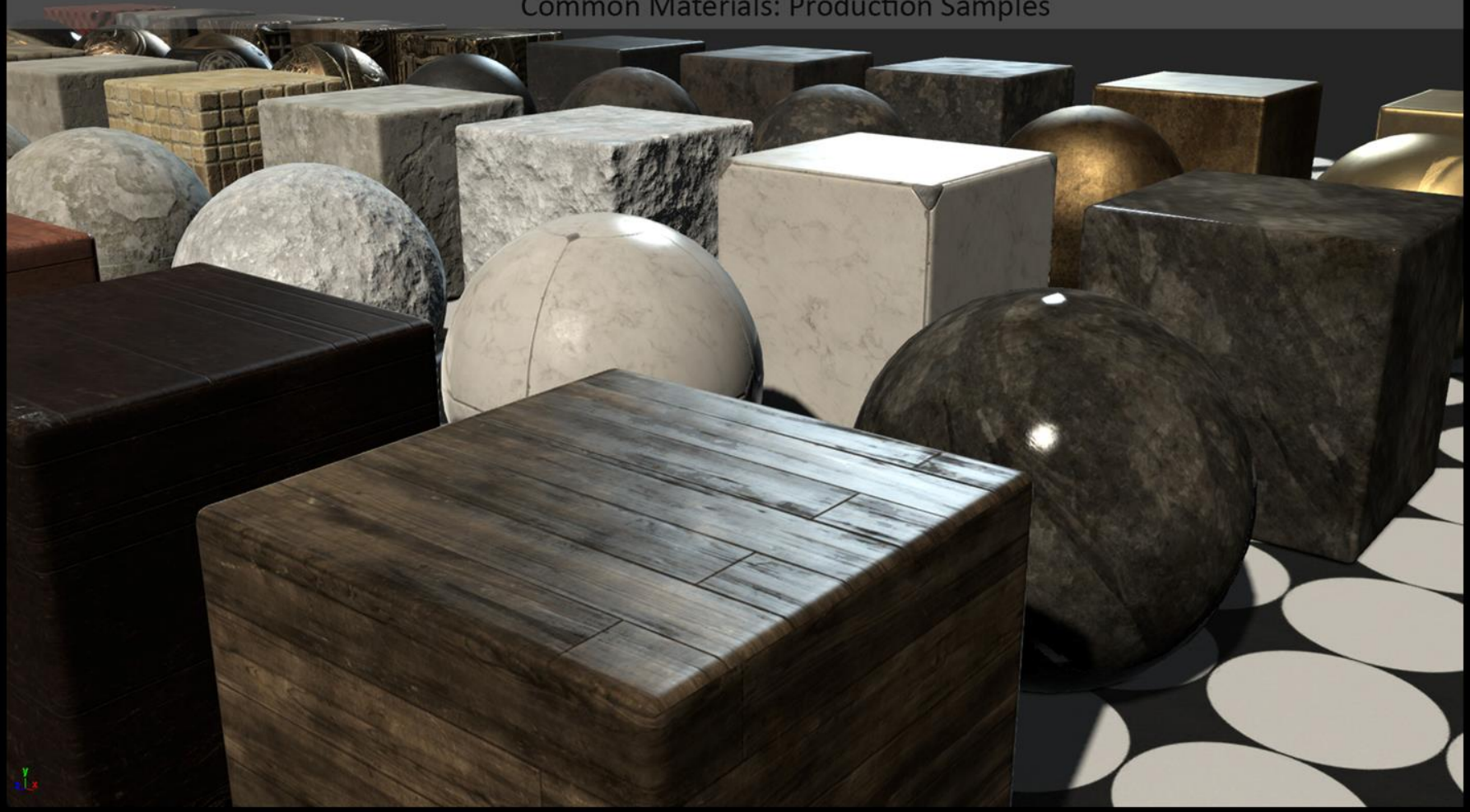
Common Materials: Production Samples



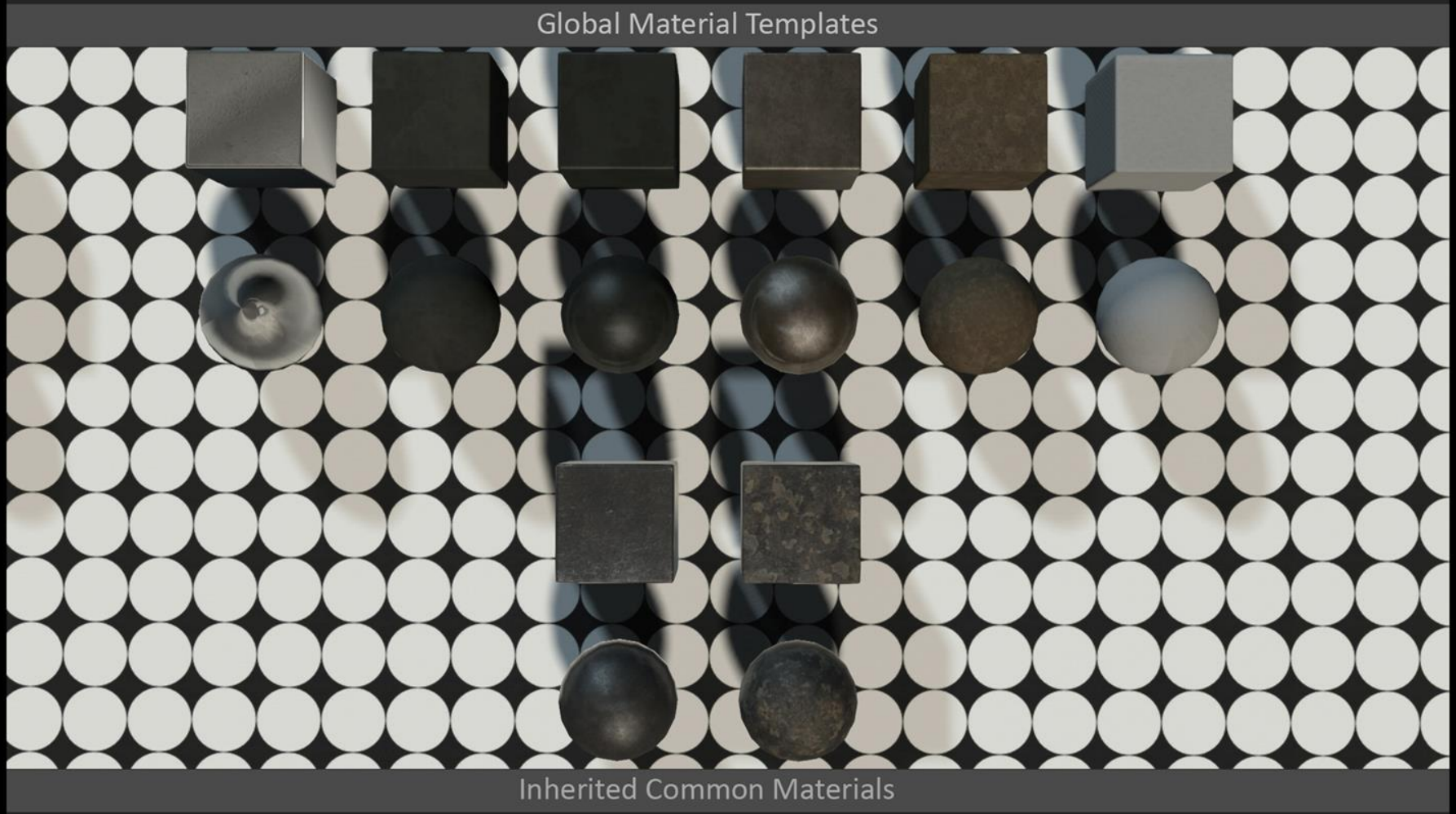
Common Materials: Production Samples

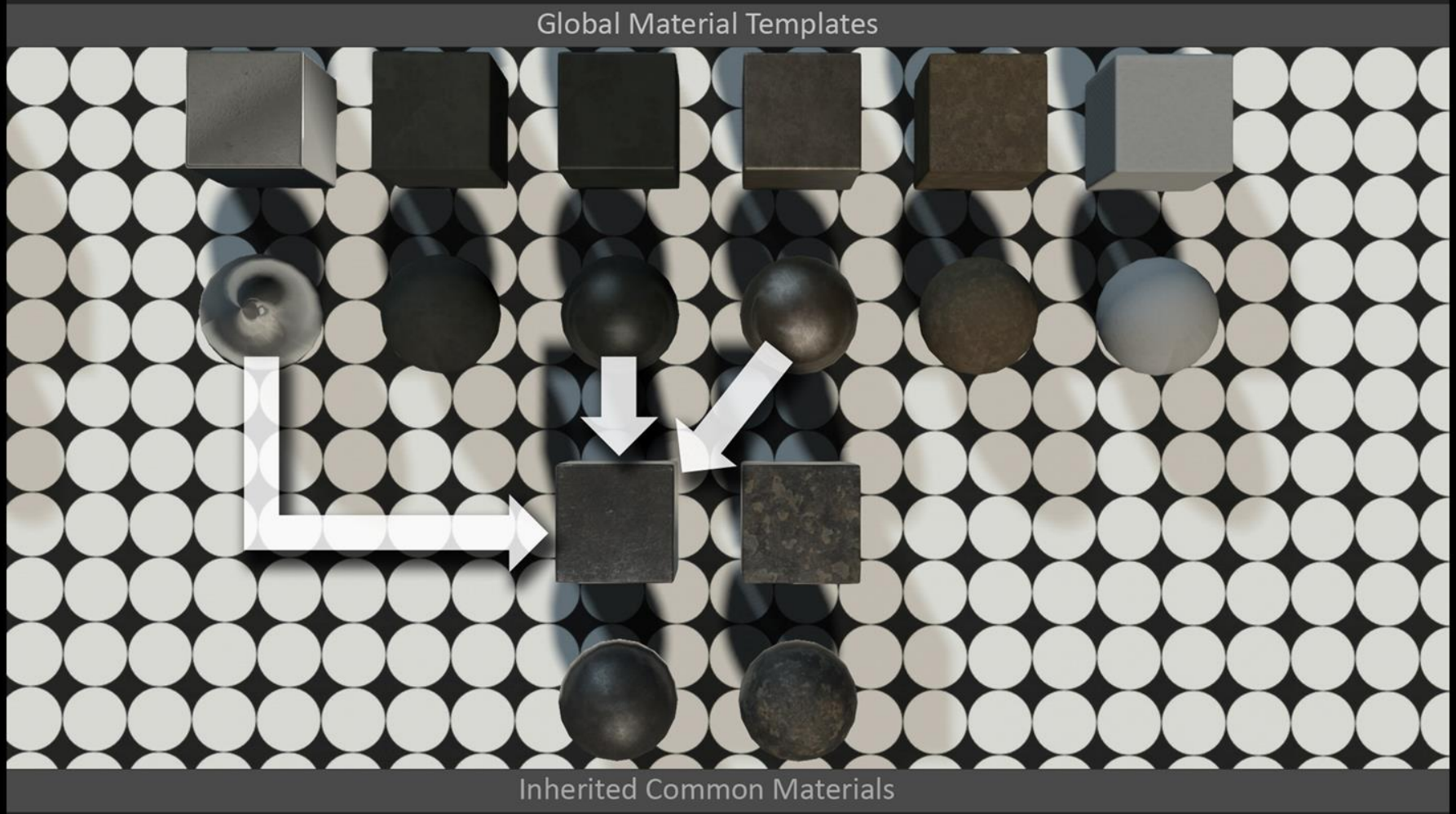


Common Materials: Production Samples



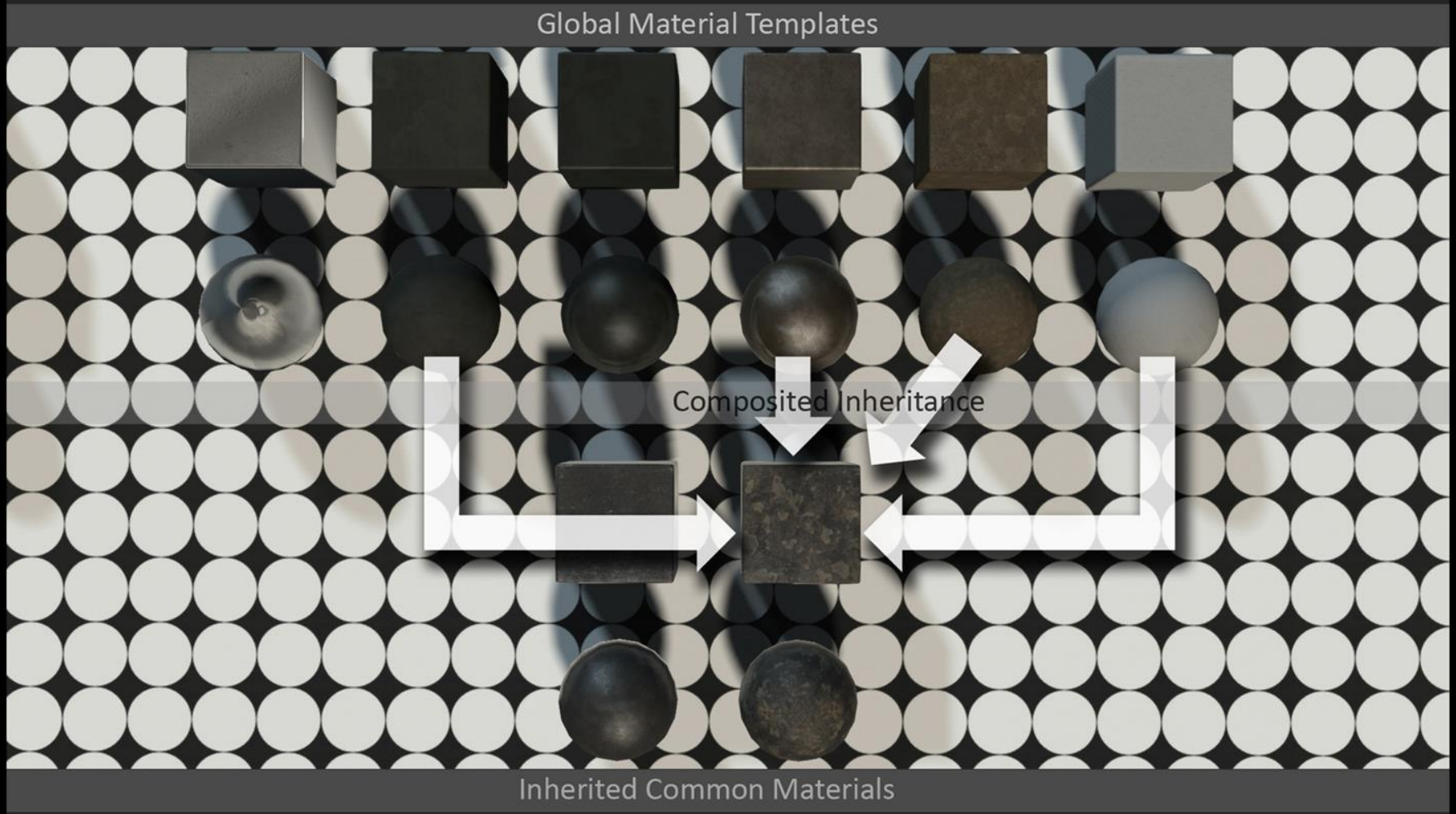
Common Materials: Production Samples


Global Material Templates
Inherited Common Materials


Global Material Templates
Inherited Common Materials

Global Material Templates

Inherited Common Materials


Global Material Templates
Composited Inheritance
Inherited Common Materials

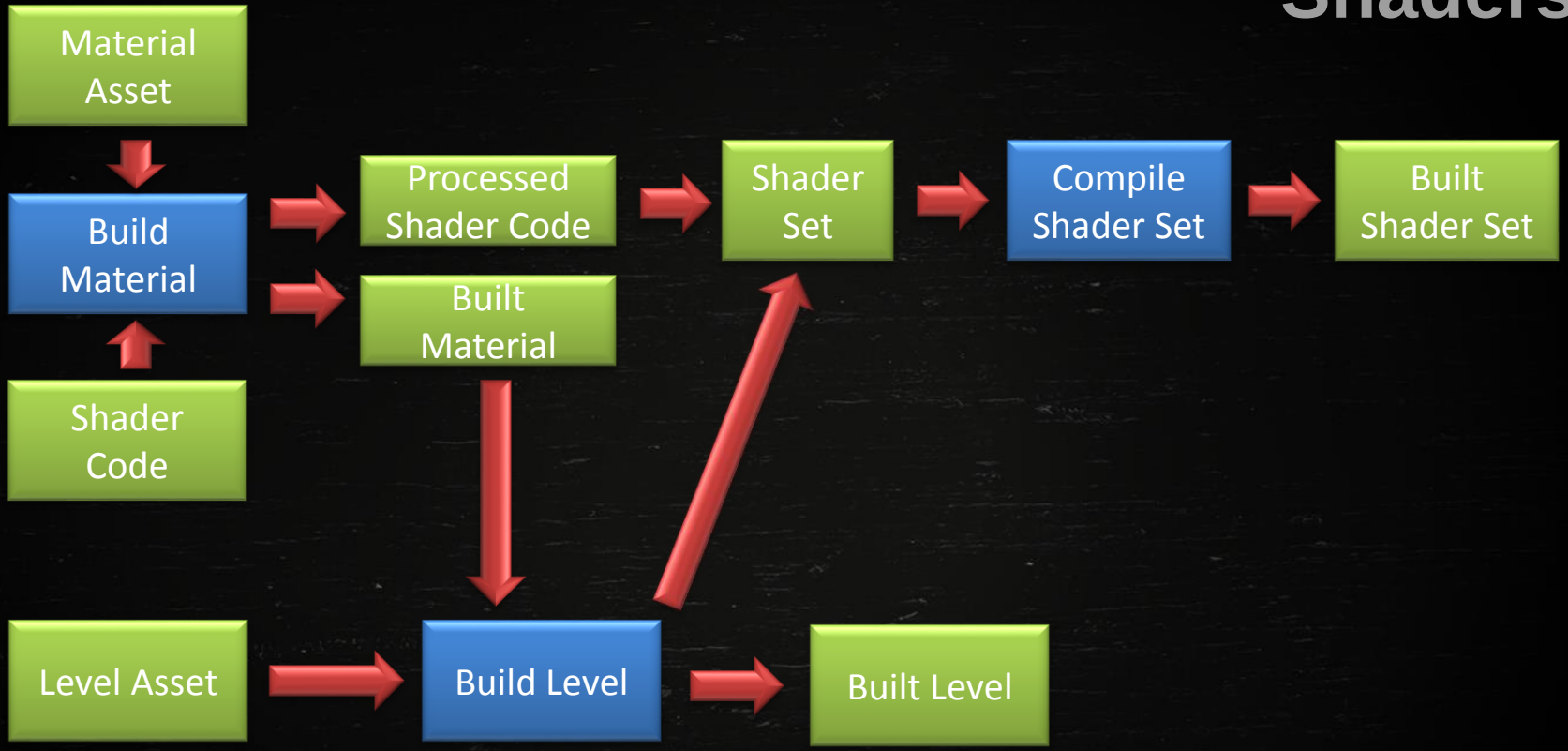
Shaders

- Hand-written ubershaders
  - Lots of #if's
- Major material features = macro definitions
- Parameters (spec intensity, roughness, etc.) hard-coded into the shader, except when animated or composited
- Some code is auto-generated to handle textures and animated parameters

Shaders

- Pre-defined “permutations”
  - Permutation = macro definition + entry point
- Permutations for skinning, blend shapes, instancing, light maps, etc.
- Debug permutation for visualization/debugging
- Shaders compiled in build pipeline based on permutation + material

Shaders



Shaders

- Pros:
  - Shaders are optimized for a material
    - Optimizer has full access
  - No runtime compiling for the game itself (used by tools)
- Cons:
  - *Lots* of shaders to compile!
    - We cache everything, but iteration can be slow
  - Monolithic shaders are hard to debug
  - Lean heavily on the compiler

Material Compositing

Material Compositing

- Mostly offline process
- Material asset specifies composite “stack”
- Each layer in the stack has:
  - Referenced material
  - Blend mask
  - Blending parameters
- Recursively build + composite referenced materials
- Combine each layer 1 by 1 using pixel shader

Material Compositing

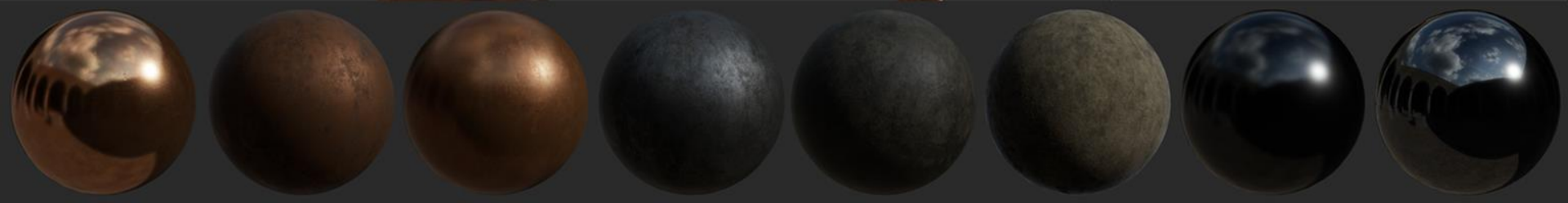
- Generates parameter maps from materials and blending maps
- Support compositing subset of BRDFs
  - Cloth, GGX, and Anisotropic
  - Compositing cloth requires applying 2 BRDFs

Material Compositing

Map	R Channel	G Channel	B Channel	A Channel	Format
1	Normals X	Normals Y	N/A	N/A	BC5
2	Diffuse R	Diffuse G	Diffuse B	Alpha (Optional)	BC1 or BC3
3	Specular R	Specular G	Specular B	Specular Intensity	BC3
4	Roughness	AO	BRDF Blend	Anisotropy	BC3

Material Compositing

```
// Compositing pixel shader run on a quad covering the entire output texture
CompositeOutput CompositePS(in float2 UV : UV) {
    CompositeOutput output;

    float blendAmt = BlendScale * BlendMap.Sample(Sampler, UV);
    float4 diffuseA = DiffuseTintA * DiffuseMapA.Sample(Sampler, UV);
    float4 diffuseB = DiffuseTintB * DiffuseMapB.Sample(Sampler, UV);
    float diffuseBlendAmt = blendAmt * DiffuseContribution;
    output.Diffuse = Blend(diffuseA, diffuseB, diffuseBlendAmt,
                           DiffuseBlendMode);

    // Do the same for specular, normals, AO, etc.

    return output;
}
```

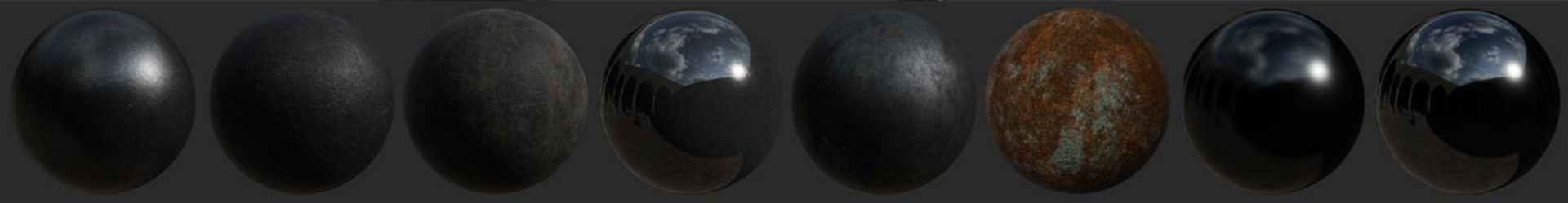


prp_well_pump_c_tarnished_copper_source
(mtt_copper_polished_075)

- BRDF
- GI Baking Properties
- Options
- Lighting
- Damage
- Skirt Blending
- Material FX
- Anisotropic Properties
- Tessellation Properties

layer0 (mtt_copper_polished_075) — Add Feature

- Surface Properties
- Diffuse Lighting
- Specular Lighting
- Composite Materials
  - Material - BASE, cmn_copper_a_tile_dark
  - Material - BASE VARIATION WORN, cmn_copper_a_tile_worn_light
  - Material - BASE VARIATION TARNISHED, cmn_tarnished_a_tile_light
  - Material - TARNISHED, cmn_steel_a_tile_worn_dark
  - Material - PRISTINE, cmn_copper_a_tile_pristine_light
  - Material - DIRT, cmn_dirt_fine_a_tile_tan_dark
  - Material - WET 50, cmn_water_dampness_050
  - Material - WET, cmn_water_dampness_100



prp_well_pump_c_worn_enamel_black_source
(cmn_enamel_fine_a_tile_pristine_black)

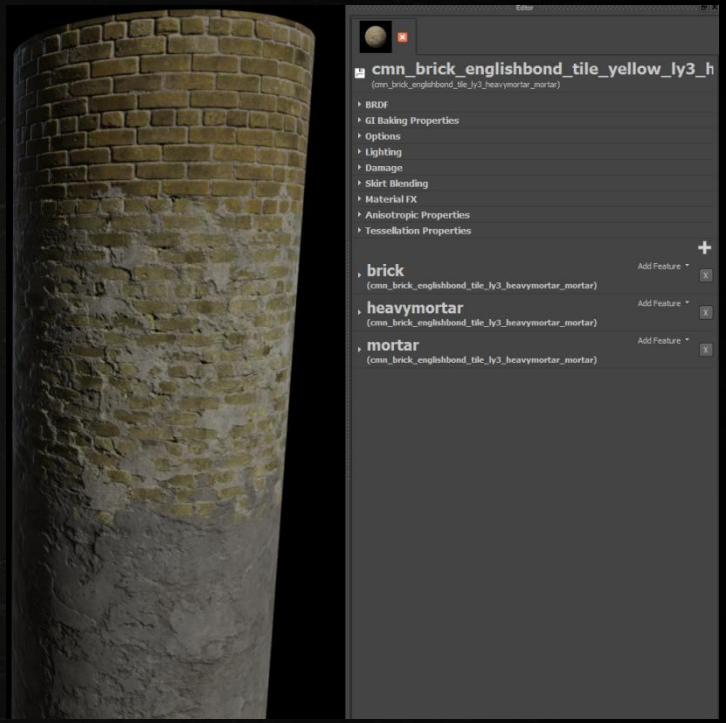
- BRDF
- GI Baking Properties
- Options
- Lighting
- Damage
- Skirt Blending
- Material FX
- Anisotropic Properties
- Tessellation Properties

layer0 (cmn_enamel_fine_a_tile_pristine_black) — Add Feature

- Surface Properties
- Diffuse Lighting
- Specular Lighting
- Composite Materials
  - Material - BASE, cmn_enamel_fine_a_tile_pristine_black
  - Material - BASE VARIATION WORN, cmn_enamel_fine_a_tile_black
  - Material - BASE VARIATION TARNISHED, cmn_enamel_fine_a_tile_wo
  - Material - TARNISHED, cmn_steel_a_tile_worn_dark
  - Material - PRISTINE, cmn_steel_a_tile_pristine_dark
  - Material - RUST, cmn_rust_coarse_a_tile_brown_mix
  - Material - WET, cmn_water_dampness_075
  - Material - WET 50, cmn_water_dampness_050

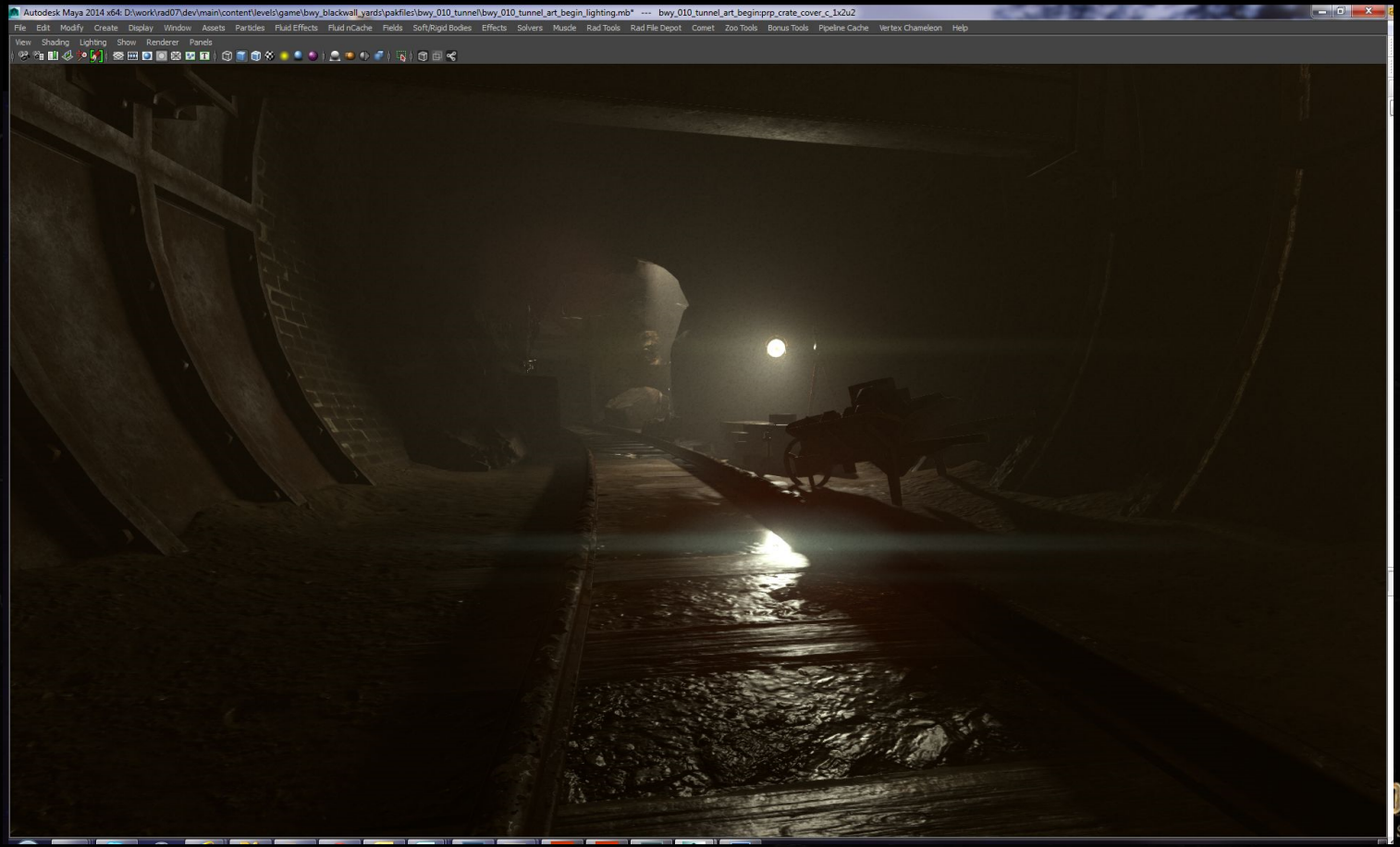
Material Layers

- Up to 4 layers
  - Derived from base materials
  - Separate compositing chain per layer
  - Driven by vertex colors

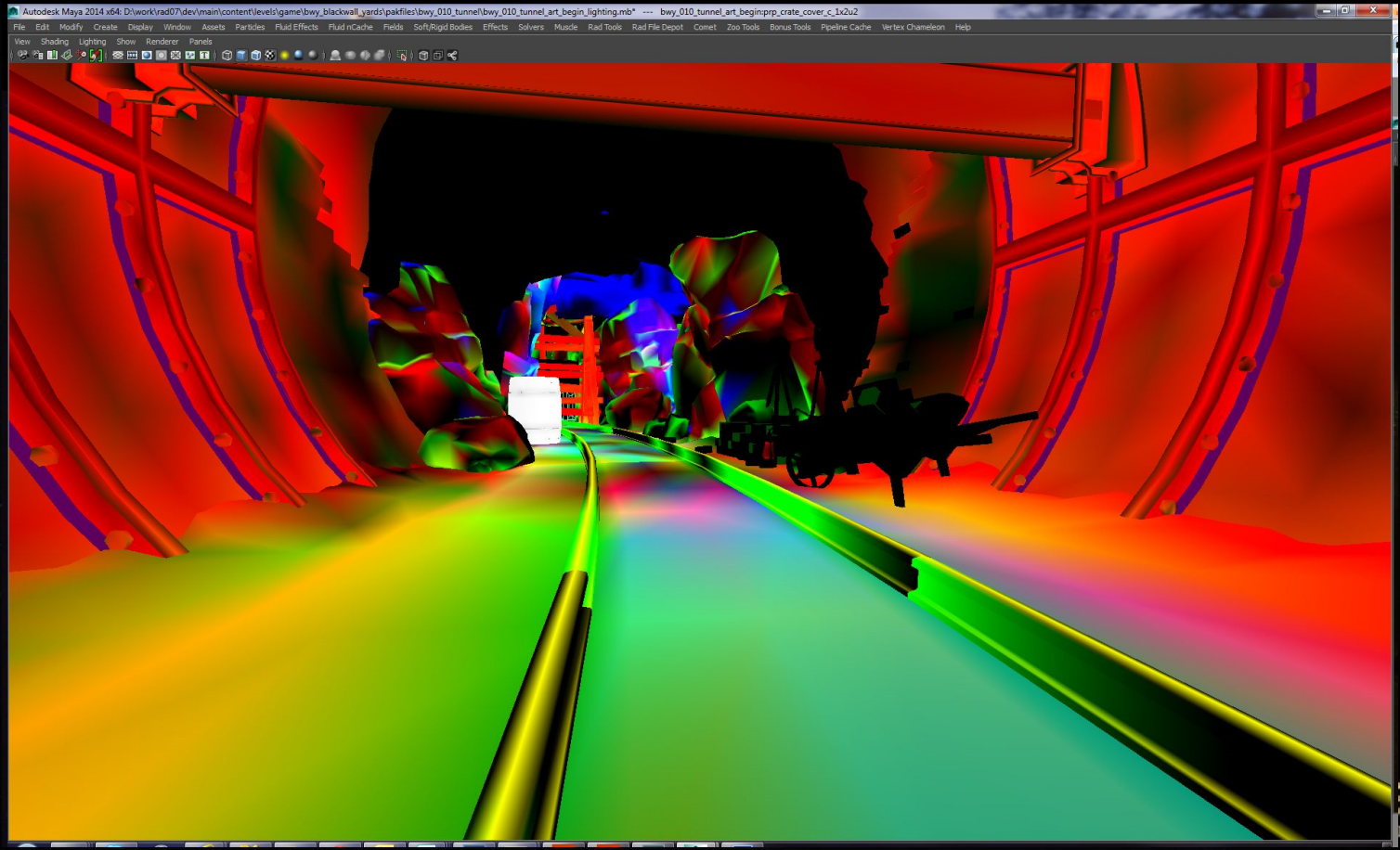


Readyatdawn STUDIOS

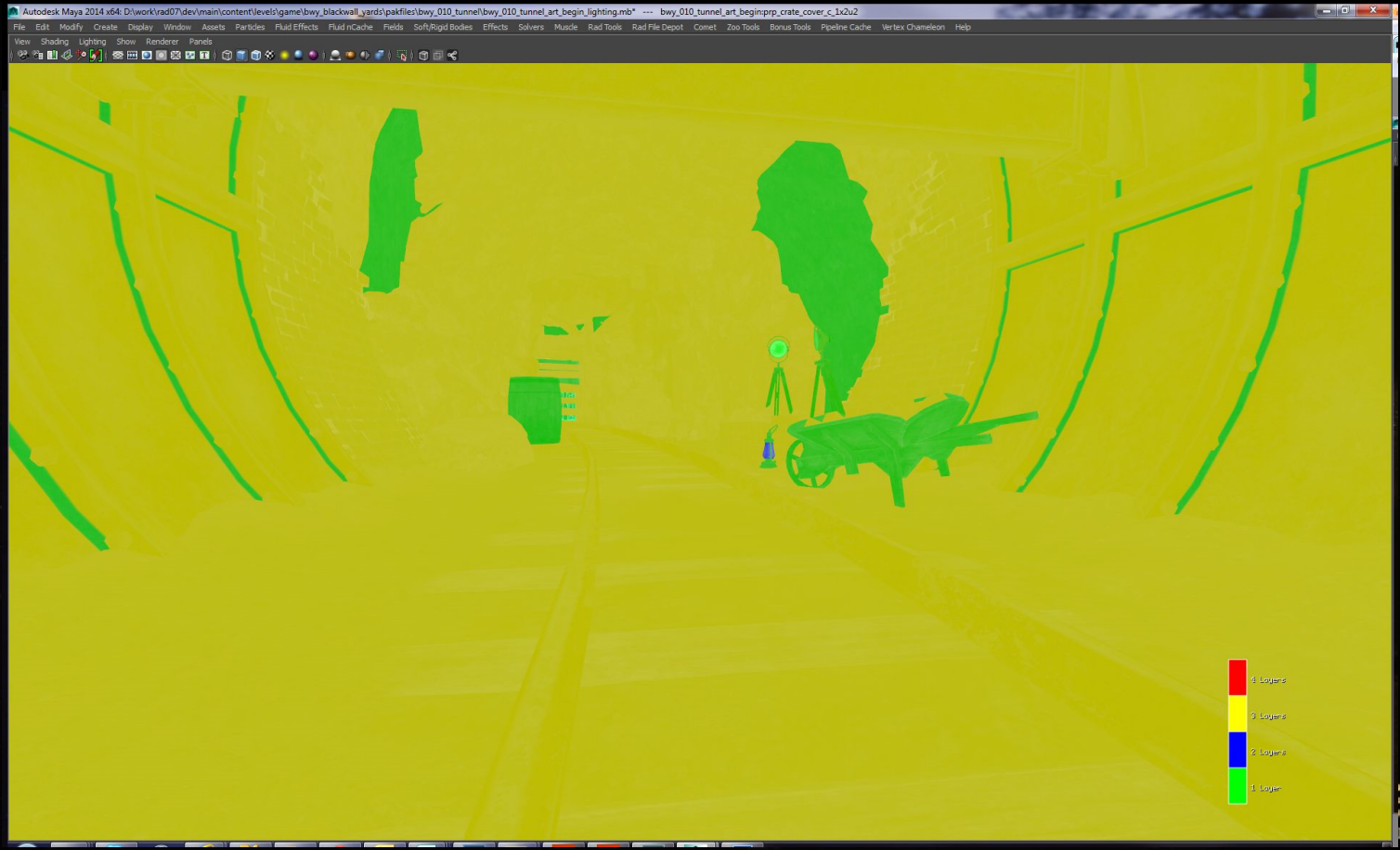
Material Layers



Material Layers



Material Layers



Material Layers

```
LayerParams combinedParams;

[unroll]
for(uint i = 0; i < NumLayers; ++i) {
    // Build all layer params from textures and hard-coded
    // material parameters
    LayerParams layerParams = GetLayerParams(i, MatParams, Textures);

    // Blend with the previous layer using vertex data and blend masks
    combinedParams = BlendLayer(combinedParams, layerParams, vtxData,
                                BlendMode, Textures);
}

// Calculate all lighting using the blended params
return ComputeLighting(combinedParams);
```

Readyatbawn
STUDIOS

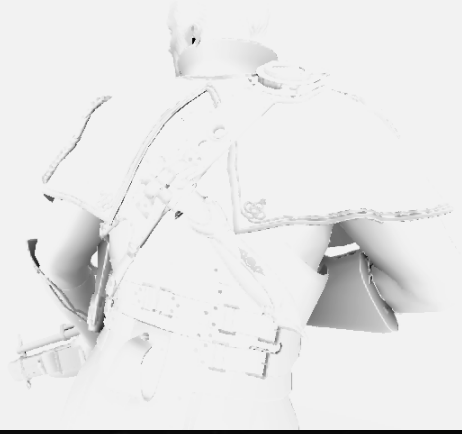
Readyatdawn
STUDIOS


Readyatbawn
STUDIOS

Readyatdawn
STUDIOS

Readyatdawn
STUDIOS

Readyatdawn
STUDIOS



Readyatdawn
STUDIOS

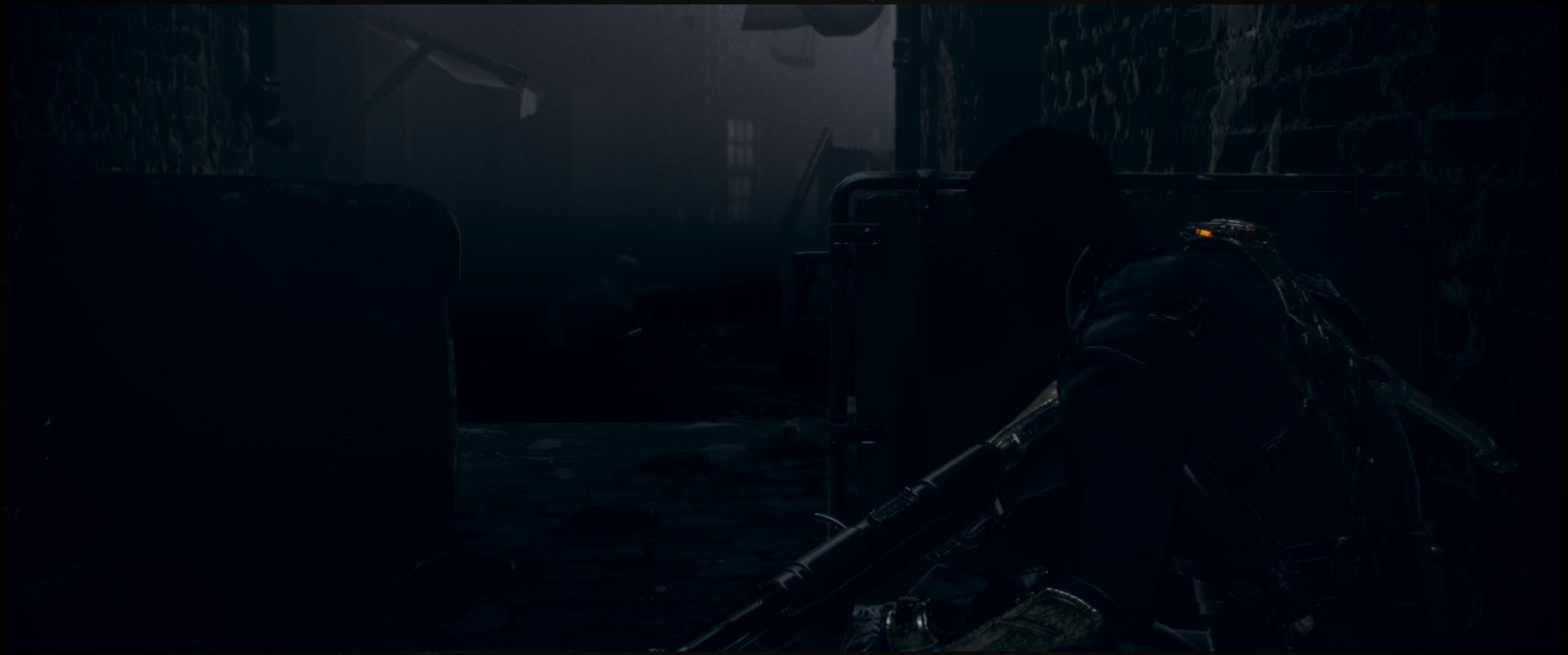
Readyatbawn
STUDIOS


Readyatbawn
STUDIOS

Readyatdawn
STUDIOS

Readyatdawn
STUDIOS

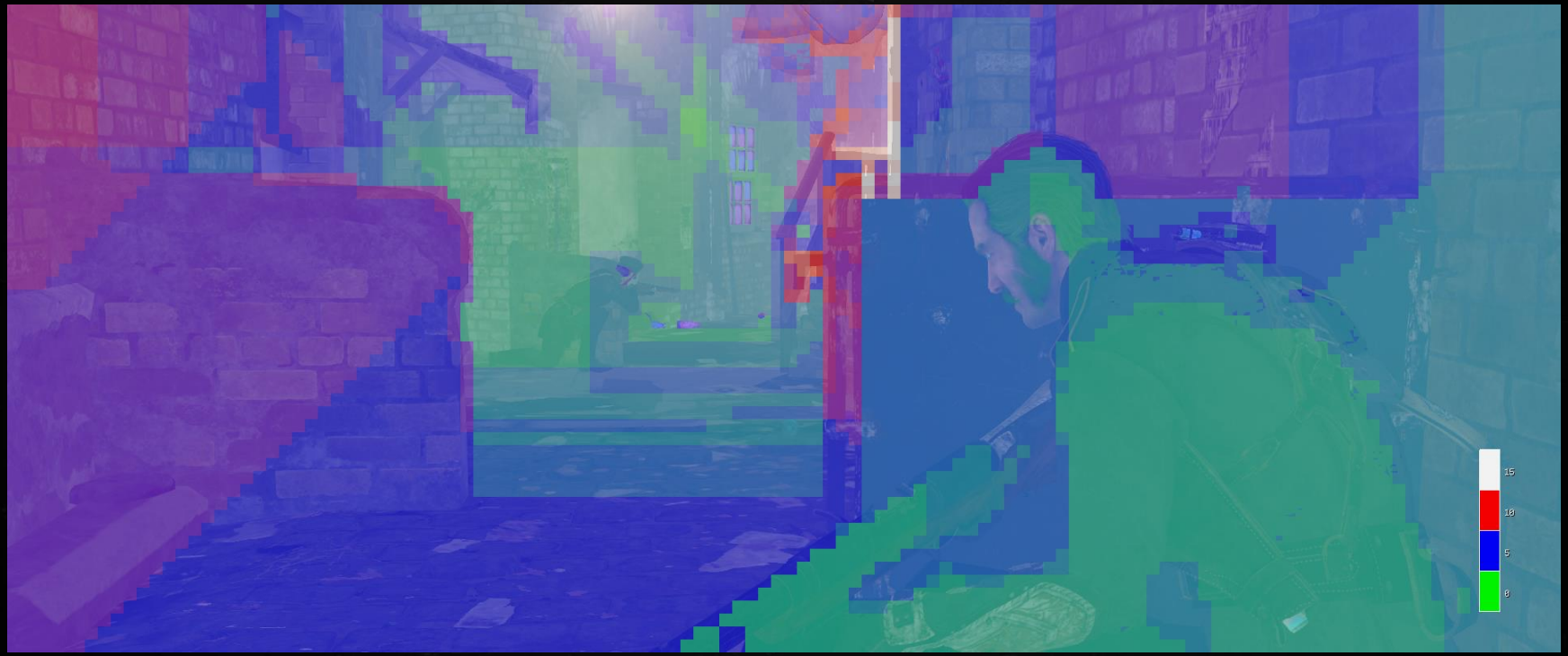
Readyatdawn
STUDIOS


Readyatbawn
STUDIOS


Readyatbawn
STUDIOS

Readyatdawn
STUDIOS

Readyatdawn
STUDIOS



Readyatdawn
STUDIOS

Readyatdawn
STUDIOS


Readyatbawn
STUDIOS

Credits

Programmers

David Neubelt
david@readyatdawn.com
twitter: @daveneubelt

Matt Pettineo
matt@readyatdawn.com
Twitter: @MyNameIsMJP
Blog: http://mynameismjp.wordpress.com/

Garret Foster
garret@readyatdawn.com

Artists

Nathan Phail-Liff
nathan@readyatdawn.com

Readyatdawn STUDIOS

Questions?