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

by David Neubelt and Matt Pettineo, Ready at Dawn Studios

Introduction

The Order: 1886 is an upcoming third-person action-adventure game for the PlayStation 4. When we first started pre-production for the project, the word “filmic” was often used to describe the cinematic feel that we wanted to achieve with the game’s visuals. Things like cinematic lighting, grungy streets, and highly detailed characters were identified as key elements in creating a convincing portrayal of 19th-century London.

It was decided very early in the project that utilizing a physically based shading model would be a key component of achieving the look we wanted for the game. A Cook-Torrance-like microfacet BRDF was adopted as the default specular BRDF for our materials, and energy conservation was enforced throughout the rendering pipeline. We also implemented several alternative BRDFs for skin, hair, fabrics, and anisotropic materials.

We also realized that having a robust material pipeline would be essential for allowing artists to efficiently create the thousands of materials that would fill our game world. To this end we utilized an inheritance-based asset format for our materials that allowed us to create a standard library of material templates that could be modified to create level-specific variations of core material types. We also implemented an offline material compositing pipeline for automatically generating parameter maps for materials, with the parameters being defined by a stack of material assets and blend maps. In addition to the offline compositing, we implemented a runtime layering component for materials that allows arbitrary materials to be combined inside the pixel shader.

To quickly add detailed high-quality materials for our characters and props, we created a 3D textile scanner. The scanner captures high resolution albedo and normal maps and from those we can derive additional textures. We integrated the maps into our material pipeline using a detail layer in our material, in order to overlay the high-frequency normal and albedo variation.

Finally, we researched and implemented a technique for reducing aliasing when directly evaluating our specular BRDFs in the pixel shader. Our technique is based on the paper *Frequency Domain Normal Map Filtering* Han et al. [2007a], which uses the frequency domain to convolve a BRDF with an NDF that's computed for the mip levels of a normal map.

Core Shading Model

Microfacet Specular BRDF

Physically based microfacet BRDFs are quickly becoming the status quo in real-time rendering due to their robustness and quality relative to ad hoc shading models, and the introduction of more powerful GPUs in next-generation console hardware is likely to accelerate that trend. Since *The Order: 1886* is releasing on the PlayStation 4, we felt that physically based BRDFs were a natural fit for the immense computational throughput of our target hardware. Our default specular BRDF follows the form of the Cook-Torrance microfacet BRDF Cook and Torrance [1982]:

$$f(\mathbf{l},\mathbf{v}) = \frac{F(\mathbf{v},\mathbf{h})\,G(\mathbf{l},\mathbf{v},\mathbf{h})\,D(\mathbf{h})}{4\,(\underline{\mathbf{n}\cdot\mathbf{l}})\,(\underline{\mathbf{n}\cdot\mathbf{v}})} \tag{1}$$

Where $\mathbf{l}$ is the light direction, $\mathbf{v}$ is the view direction, $\mathbf{h}$ is the half-vector, $\mathbf{n}$ is the normal, F is the Fresnel term[1], G is the geometry term[2], and D is the normal distribution function (NDF). For the D term we use the GGX distribution presented in Walter et al. [2007], and for the G term we use the matching Smith shadowing term derived in the same paper:

$$D(\mathbf{m}) = \frac{\alpha^2}{\pi\left((\underline{\mathbf{n}\cdot\mathbf{m}})^2\,(\alpha^2-1)+1\right)^2} \tag{2}$$

$$G_1(\mathbf{n},\mathbf{v}) = \frac{2}{1+\sqrt{1+\alpha^2\,(1-(\underline{\mathbf{n}\cdot\mathbf{v}})^2)/(\underline{\mathbf{n}\cdot\mathbf{v}})^2}} \tag{3}$$

$$G(\mathbf{l},\mathbf{v},\mathbf{h}) = G_1(\mathbf{n},\mathbf{l})\,G_1(\mathbf{n},\mathbf{v}) \tag{4}$$

Where $\mathbf{m}$ is the active microfacet normal direction, and α is the roughness parameter. In practice the BRDF can be simplified via the following rearrangement of G_1:

$$G_1(\mathbf{n},\mathbf{v}) = \frac{2\,(\underline{\mathbf{n}\cdot\mathbf{v}})}{(\underline{\mathbf{n}\cdot\mathbf{v}})+\sqrt{1+\alpha^2\,(1-(\underline{\mathbf{n}\cdot\mathbf{v}})^2)}} \tag{5}$$

$$= \frac{2\,(\underline{\mathbf{n}\cdot\mathbf{v}})}{(\underline{\mathbf{n}\cdot\mathbf{v}})+\sqrt{\alpha^2+(1-\alpha^2)\,(\underline{\mathbf{n}\cdot\mathbf{v}})^2}} \tag{6}$$

[1] As is common, we use Schlick's Approximation Schlick [1994] here.
[2] Also known as the shadowing-masking term.

Similar to Lazarov [2011] we can then make the following substitution, introducing V, a *visibility* term:

$$f(\mathbf{l},\mathbf{v}) = F(\mathbf{v},\mathbf{h})\,V(\mathbf{l},\mathbf{v},\mathbf{h})\,D(\mathbf{h}) \tag{7}$$

$$\text{where } V(\mathbf{l},\mathbf{v},\mathbf{h}) = \frac{G(\mathbf{l},\mathbf{v},\mathbf{h})}{4\,(\underline{\mathbf{n}\cdot\mathbf{l}})\,(\underline{\mathbf{n}\cdot\mathbf{v}})} = V_1(\mathbf{n},\mathbf{l})\,V_1(\mathbf{n},\mathbf{v}) \tag{8}$$

$$\text{and } V_1(\mathbf{n},\mathbf{v}) = \frac{1}{(\underline{\mathbf{n}\cdot\mathbf{v}}) + \sqrt{\alpha^2 + (1-\alpha^2)\,(\underline{\mathbf{n}\cdot\mathbf{v}})^2}} \tag{9}$$

Typically a geometry term must be constructed such that $G(\mathbf{n},\mathbf{l}) = 0$ when $\mathbf{n}\cdot\mathbf{v}$ is less than or equal to 0 in order for it to be considered physically plausible Walter et al. [2007]. This constraint is meant to ensure that light cannot reflect off surfaces that are not visible to the viewer. Satisfying this requirement would require modifying the G term:

$$G(\mathbf{l},\mathbf{v},\mathbf{h}) = \begin{cases} G_1(\mathbf{n},\mathbf{l})\,G_1(\mathbf{n},\mathbf{v}) & \text{if } \mathbf{n}\cdot\mathbf{v} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

In practice we have found that pixels with $\mathbf{n}\cdot\mathbf{v} \le 0$ occur quite frequently when normal maps are used. If rendered with displacement mapping, these surfaces would be occluded after rasterization, but with conventional normal-mapping techniques these surfaces remain visible and must be shaded. While it would be possible to shade such pixels with the previously described G term, doing so results in visual artifacts due to the discontinuity at $\mathbf{n}\cdot\mathbf{v} = 0$. Instead, we simply shade these pixels without the conditional as if they were front-facing. Doing so allows the grazing-angle specular to fall off on the side facing away from the viewer, which produces an artifact-free result.

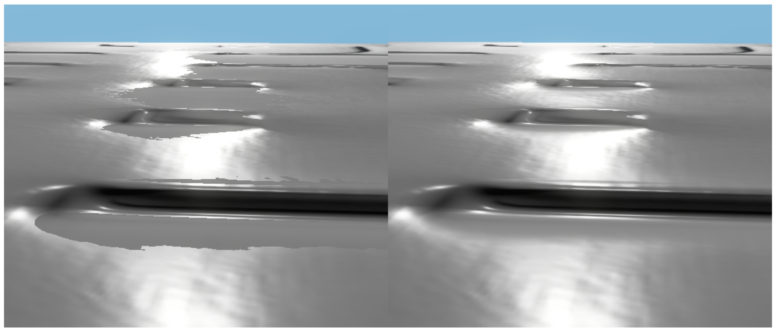


Figure 1: The left image shows artifacts resulting from using a conditional to force the geometry term to 0 when $\mathbf{n}\cdot\mathbf{v} \le 0$. The right image shows the same scene without the conditional.

Anisotropic Specular BRDF

To handle anisotropic materials we use the substitution:

$$\frac{1}{\alpha^2} = \frac{\cos^2\theta}{{\alpha_t}^2} + \frac{\sin^2\theta}{{\alpha_b}^2} \tag{11}$$

where α_t is the roughness along tangent direction, and α_b is the roughness along the bitangent direction.

Instead of exposing two roughness parameters, our material model utilizes a single roughness parameter α in conjunction with an anisotropy parameter that describes the relationship between the two roughness values:

$$\alpha_t = \alpha \tag{12}$$

$$\alpha_b = \mathrm{lerp}(0, \alpha, 1 - \mathit{anisotropy}) \tag{13}$$

Using math substitutions from Burley [2012], we come to the optimized form[3] used for shading:

$$D_{aniso}(\mathbf{m}) = \frac{1}{\pi\,\alpha_t\,\alpha_b} \frac{1}{((\frac{\mathbf{t}\cdot\mathbf{m}}{\alpha_t})^2 + (\frac{\mathbf{b}\cdot\mathbf{m}}{\alpha_b})^2 + (\underline{\mathbf{n}\cdot\mathbf{m}})^2)^2} \tag{14}$$

Diffuse BRDF

For the diffuse component, we use a simple Lambertian diffuse model. In order to ensure energy conservation, the diffuse term is balanced using the inverse of the Fresnel term from the specular component Shirley [1991]:

$$f(\mathbf{l},\mathbf{v}) = (1 - F(\underline{\mathbf{v}\cdot\mathbf{h}}))\,\frac{\mathbf{c_{diff}}}{\pi} \tag{15}$$

Where $\mathbf{c_{diff}}$ is the diffuse albedo of the material. It should be noted that balancing the diffuse term in this manner violates Helmholtz reciprocity, which can cause issues with certain rendering techniques. If reciprocity is desired, an alternative balancing scheme is proposed in Shirley et al. [1997] that satisfies reciprocity at the cost of additional instructions.

Diffuse BRDF for Skin

To simulate sub-surface scattering effects present in skin, we utilize the pre-integrated skin shading technique presented in Penner and Borshukov [2011]. This technique approximates sub-surface scattering effects by pre-integrating a skin diffusion profile $R(x)$ convolved with a Lambertian diffuse BRDF for various points on a ring with radius r:

$$D(\theta, r) = \frac{\int_{-\pi}^{\pi} \cos(\theta + x) R(2r\sin(x/2))dx}{\int_{-\pi}^{\pi} R(2r\sin(x/2))dx} \tag{16}$$

Where θ is the angle between a given point on the ring and the incident lighting direction $\mathbf{l}$. The result of this pre-integration can then be used as a replacement for the standard clamped cosine transfer function when determining the diffuse reflectance of a surface with curvature equal to a given radius r. By performing the integral for a certain range of r, a 2D lookup texture is formed that contains an array of such transfer functions. For convenience this texture is indexed by $\cos(\theta)$ instead of θ to avoid expensive inverse trignometric functions. This makes for trivial evaluation of punctual light sources:

$$f_{skin}(\mathbf{n},\mathbf{l}) = D(\mathbf{n}\cdot\mathbf{l}, r)L_i \tag{17}$$

Where L_i is the intensity of the light source. While this approach worked well for light sources evaluated directly in the pixel shader, our engine also utilizes ambient light probes that store a lighting environment as 3rd-order spherical harmonics. Initially, these light probes were evaluated using a simple Lambertian diffuse BRDF, which resulted in our skin taking on a hard, unnatural look in environments without strong direct lighting, see Figure 2.

To improve the shading quality in these areas we implemented a solution for utilizing the pre-integrated scattering with a spherical harmonics lighting environment. Recall from Ramamoorthi et al. [2001] that the irradiance E incident on a surface due to lighting environment L can be represented as a frequency-domain convolution using spherical harmonics:

$$E_{l,m} = \sqrt{\frac{4\pi}{2l+1}} A_l L_{l,m} \tag{18}$$

Where A_l represents the transfer function, which in the case of irradiance is the clamped cosine function. This derivation takes advantage of the fact that the clamped cosine term is radially symmetric,

[3]Where $\mathbf{t}$ and $\mathbf{b}$ are the surface tangent and bitangent, respectively.

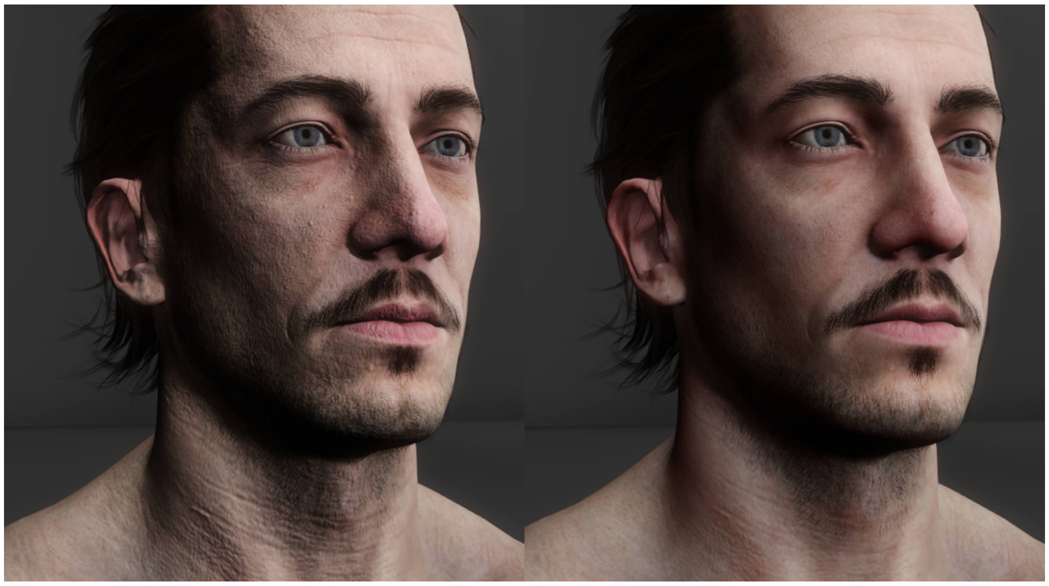


Figure 2: Images showing a skin material being lit by a diffuse SH probe only (no specular). The left image shows the convolution of the SH lighting environment with a standard Lambertian cosine lobe. The right image shows our solution, which convolves the lighting environment with a lobe generated by pre-integration of a skin diffusion profile.

which allows use of zonal harmonics in representing the transfer function as well as simplified rotations for transforming the lighting environment from global space to a coordinate space local to the surface normal. Since our skin scattering transfer function is also radially symmetric, we can compute zonal harmonics coefficents $D_l(r)$ for this function and use them to replace the clamped cosine term. Computation of these coefficents is done by integrating the transfer function $D(\theta, r)$ against the spherical harmonics basis functions $Y_{l,m}$ up to the desired order, which is 3 in our case[4]:

$$D_n(r) = 2\pi \int_0^{\pi/2} Y_{n,0}(\theta) D(\theta, r) sin(\theta) d\theta \tag{19}$$

Since we do not have an analytical representation of $D(\theta, r)$, we estimate the result of the integral using Monte Carlo integration. This integration is performed for our supported range of curvatures, producing a 1D lookup table $D_l(r)$ that can be indexed at runtime using surface curvature. Computing the diffuse reflectance for a surface with a normal $\mathbf{n}$ and curvature r can then be performed as demonstrated in Ramamoorthi et al. [2001]:

$$f_{skinSH}(\mathbf{n}, r) = \sum_{l=0}^{2} \sum_{m=-1}^{l} \sqrt{\frac{4\pi}{2l+1}} D_l(r) L_{l,m} Y_{l,m}(n) \tag{20}$$

Cloth Shading

For cloth, our artists initially tried to make high quality materials with our default microfacet BRDF but could never get convincing results. Part of the problem is that the microfacet model uses the underlying assumption that a surface is constructed of random v-shaped grooves that behave as perfect Fresnel mirrors. However, careful inspection of our textile library under a magnified camera lens, showed that cloth follows a microfiber model breaking that assumption. Using a textile library we've collected, e.g. Figure 4 and Figure 5, we identified that cloth needed a handful of properties:

[4]Specifically, we use 3 coefficients for each color channel giving a total of 9 scalars

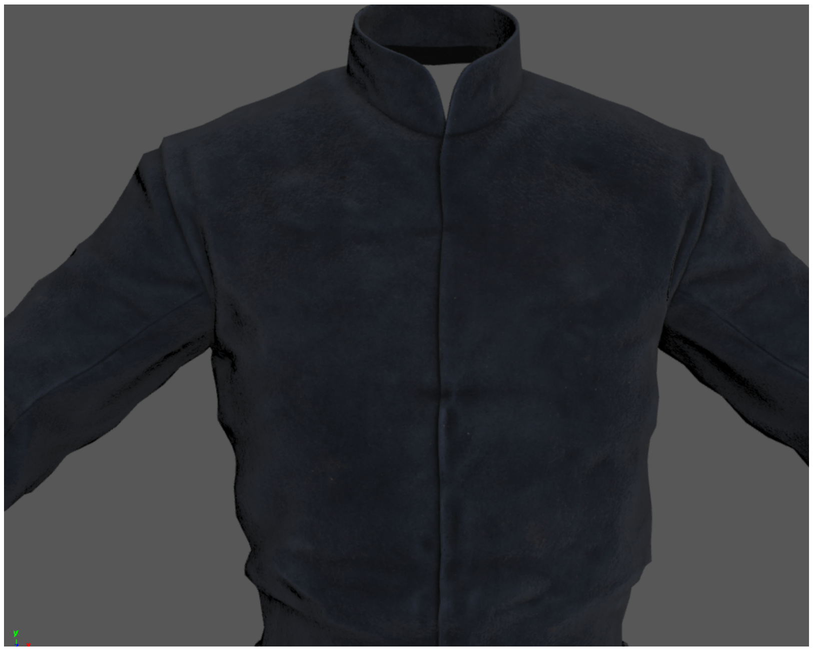


Figure 3: Example of a simple cloth jacket

- Soft specular lobes with large smooth falloffs
- Fuzz on the rim of objects from asperity scattering, either from forward scattering when the light is behind the object, or from backward scattering when the light and view direction are the same.
- A low but noticeable specular contribution at front facing angles (light, view and normal are the same)
- Some fabrics have two tones for their specular color

For fabrics, like black velvet, the most distinguishing features are due to rim lighting (both forward and backward scattering). If the light is in the same direction as the viewer then specular contributes most towards the edge of the object due to backscattering and how the fabric is constructed. Tiny fibers are attached to the surface so that they try to stand up straight. When the light and view direction are aligned the light will backscatter when the surface normal is 90 degrees from the light or view direction. Additionally, if the light is behind the objects the fibers will forward scatter light through giving a nice rim light effect.

We noticed even though cloth fibers mostly want to stand up on its surface there is small random variance of the fibers. You can also force the fibers in a certain direction by grooming it with your finger. The random variance catches enough specular for non glancing angles that it was important to support in our model.

Although we model asperity scattering and front facing specular we do not model the asymmetry in the lobe shown in Figure 5. This asymmetry likely comes from the fibers not exactly standing straight up and having a tendency to lean slightly in one direction. Skewing the distribution function could achieve this effect but it was not an important quality we were looking for so we didn't pursue it.

After surveying the existing solutions, we based our work off the velvet distribution from Ashikhmin's Distribution based BRDF Ashikhmin and Premoze [2007], as it fit our requirements, with a few modifications that allows our cloth BRDF to be useful outside of velvet.

The fuzz on the edge of objects can be emulated with an inverted Gaussian (21). The distribution is closely related to the Beckmann distribution in form. Inverting the Gaussian distribution achieves

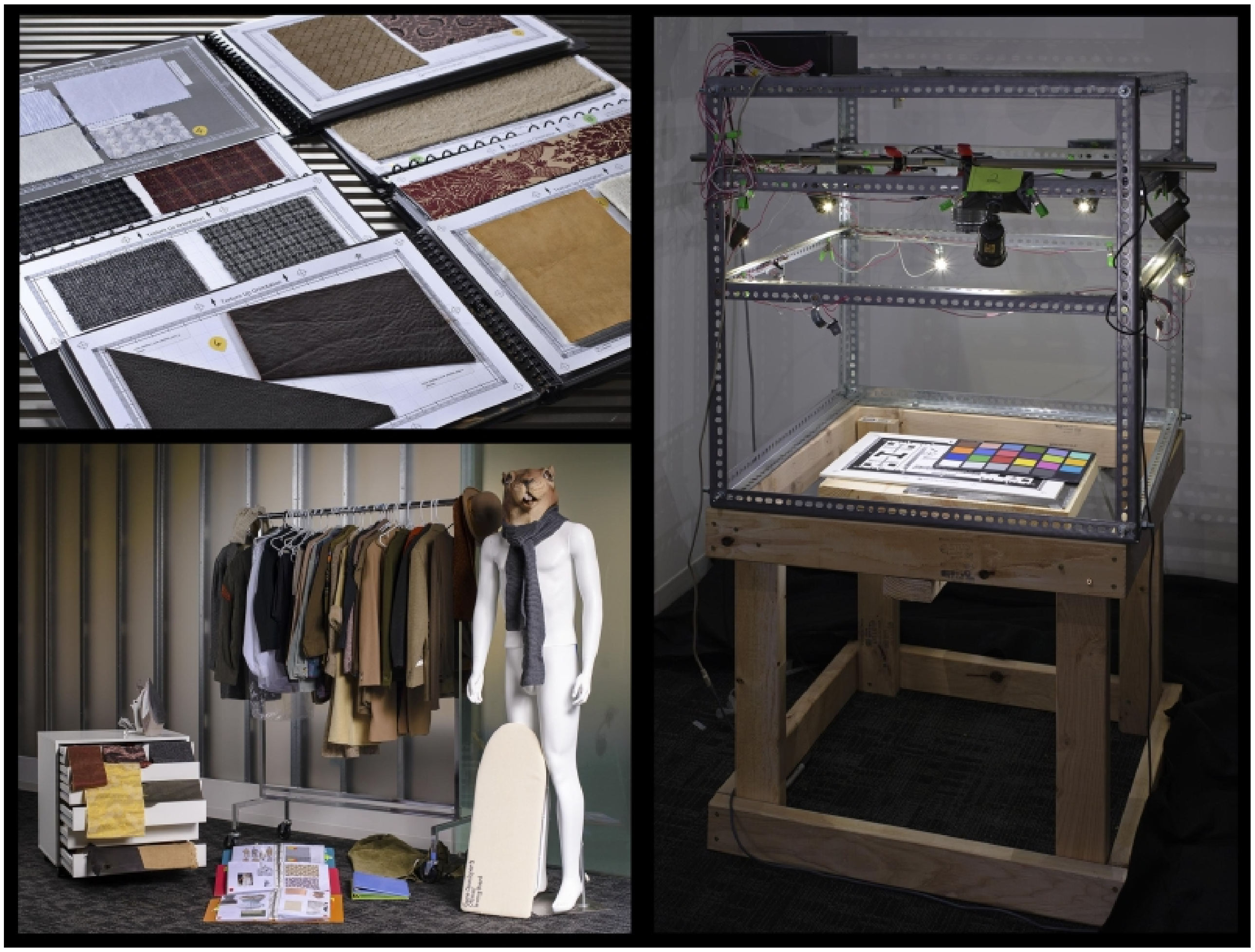


Figure 4: The bottom left image is period accurate clothing samples from the same time period the game takes place. In the top left, we take the clothing samples, cut, categorize and mount them on a page in one of our sample books. Finally, the scanner in the right scans the sample to be put into the game.

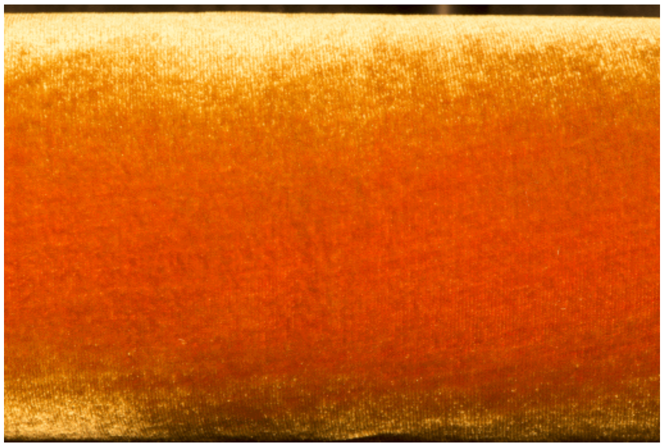


Figure 5: Two-toned velvet wrapped around a cylinder with the camera and light facing the same direction. Notice the specular color is brighter towards the edge, due to asperity scattering, and the asymmetries of the specular.

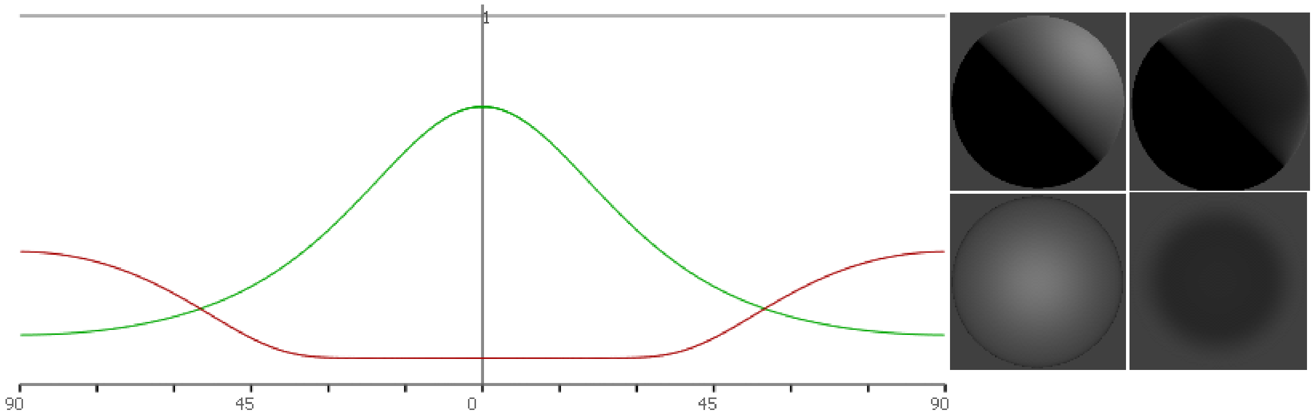


Figure 6: On the left the green line is the GGX distribution with its peak when the half angle is at 0 where the peak for the velvet distribution is when the half angle is towards 90 degrees. The middle column is a sphere using GGX specular and the right column is a sphere using the velvet specular.

the asperity scattering sheen. To get the front facing specular the specular lobe is translated off of the origin. This offset achieves that look and is similar to a diffuse term that is folded into the specular lobe and balanced.

$$D_{invgauss}(\mathbf{v}, \mathbf{h}, \alpha) = c_{norm}(1 + A \exp\left(\frac{-\cot^2\theta_h}{\alpha^2}\right)) \tag{21}$$

Equation (21) is the basic form of the velvet distribution. The A in (21) is the amplitude of the distribution which we follow Ashikhmin and use 4, 1 is the offset, and the cot is the inverse of the tan in Beckmann's distribution.

The normalization term follows closely to Beckmann's with an inversion of cos to sin and an additional step to normalize the offset.

$$D_{invgauss}(\mathbf{v}, \mathbf{h}, \alpha) = \frac{1}{\pi(1 + A\alpha^2)}(1 + \frac{A \exp\left(\frac{-\cot^2\theta_h}{\alpha^2}\right)}{\sin^4\theta_h}) \tag{22}$$

For the geometry term, we initially tried the Smith shadowing term but we found that it would darken the glancing angle specular too much. Ashikhmin noted that the distribution term is what contributes the most to a BRDF followed by the Fresnel term and finally the geometry term. He went so far to say the shadowing/masking isn't necessarily and fit his velvet distribution without any geometry term. We find this advice to be good since the fibers of velvet are partially transparent and scattering is the important effect not the shadowing and masking from v-shaped Fresnel mirrors. We want the fibers to reflect the most at the glancing angle so, like Ashikhmin, we dropped the geometry term.

The full BRDF includes a diffuse term as well to help give more material varieties then velvet such as suede and cotton. Equation (23) also replaces the traditional microfacet BRDF denominator with a smoother version, $4\,(\mathbf{n}\cdot\mathbf{l}+\mathbf{n}\cdot\mathbf{v}-(\mathbf{n}\cdot\mathbf{l})\,(\mathbf{n}\cdot\mathbf{v}))$, that isn't as hot as the traditional $4\,(\mathbf{n}\cdot\mathbf{l})\,(\mathbf{n}\cdot\mathbf{v})$, giving what we thought to be much more pleasing results.

$$(1-F(\mathbf{v}\cdot\mathbf{h}))\,\frac{\mathbf{c_{diff}}}{\pi}+\frac{F(\mathbf{v},\mathbf{h})\;D_{velvet}(\mathbf{h})}{4\,(\mathbf{n}\cdot\mathbf{l}+\mathbf{n}\cdot\mathbf{v}-(\mathbf{n}\cdot\mathbf{l})\,(\mathbf{n}\cdot\mathbf{v}))} \tag{23}$$

Cloth Future Work

For ambient specular we stored a separate set of specular probes that was convolved with the cloth BRDF. However, we also needed to modify the Fresnel curve with an offset and scalar that was dependent on $\mathbf{n}\cdot\mathbf{v}$ to make it visually match the direct lighting model. Since much of the cloths look comes from glancing angle specular and back scattering we'd like to spend a bit more time finding a closer fit.

3D Textile Scanner

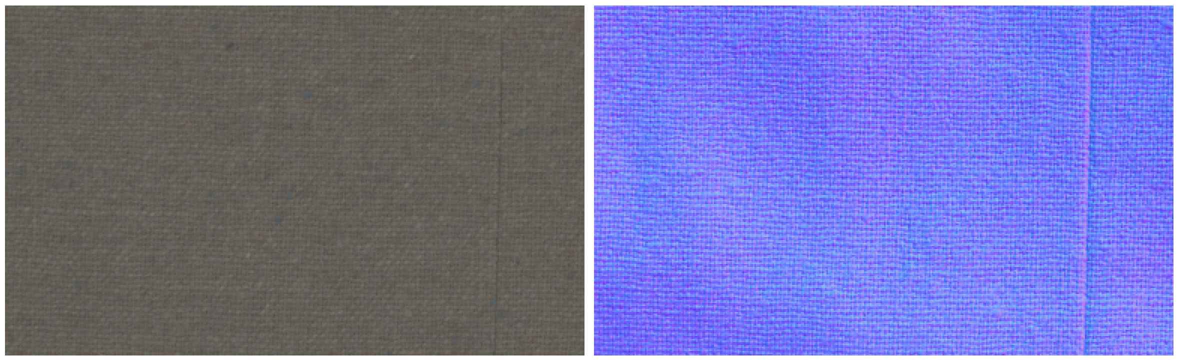


Figure 7: Zoomed in pictures of a scanned fabric sample (albedo and normal). Almost all the lighting information is removed from the albedo map. The normal map retains both the high-freqeuncy micro detail of the fabric and the larger wavey details from folds in the fabric.

The goal in designing the scanner was to provide an in-house low-cost solution for quickly getting high-frequency details and color from textiles we acquired into our engine. We did investigation into prebuilt solutions but most didn't fulfill our requirements or were too expensive to consider. All parts, outside the camera, are fairly low cost and simple to assemble with very little electronics knowledge.

The scanner uses a technique developed originally by Woodham Woodham [1980] called *Photometric Stereo* that uses different lighting conditions to reconstruct a normal vector field and color map.

A few other approaches we investigated before settling on photometric stereo were using off the shelf 3D scanners that were either based on lasers for triangulation or structured light scanning. The disadvantage of these methods usually come in the form of high-frequency noise and holes in the mesh it produces. However, with enough money you can purchase very accurate scanners or post-process (via smoothing) and remove the noise. For our use case, the high-frequency noise severely hurts the quality of detail normal maps. Photometric Stereo, on the other hand, captures high-frequency detail very well but has low-frequency errors due to inaccuracies in the measured light directions and intensities. These errors show in the maps as slow slope errors toward the edge of the image. Post-processing the normal map to become flatter is easy to do and is non-destructive to the details in the normal map.

The hardware we developed is a simple apparatus consisting of a camera and a series of D65 white LEDs. The basic idea is a calibrated camera is placed above the sample at a known position and orientation. For each light, which is also calibrated and at a known position, is turned on one at a time while taking a picture. The lights and camera are controlled by software on the computer, which communicates with a microcontroller used to control the lights. The lights' static nature allow us to calibrate for direction and intensity. From all this information we are able to solve an over determined set of 10, $(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})ik_d = \mathbf{I}$, linear equations using least squares. The software outputs a normal and albedo map. Technically, only two lights are required to solve but the additional lights gives us redundant data for filtering.

Hardware configuration

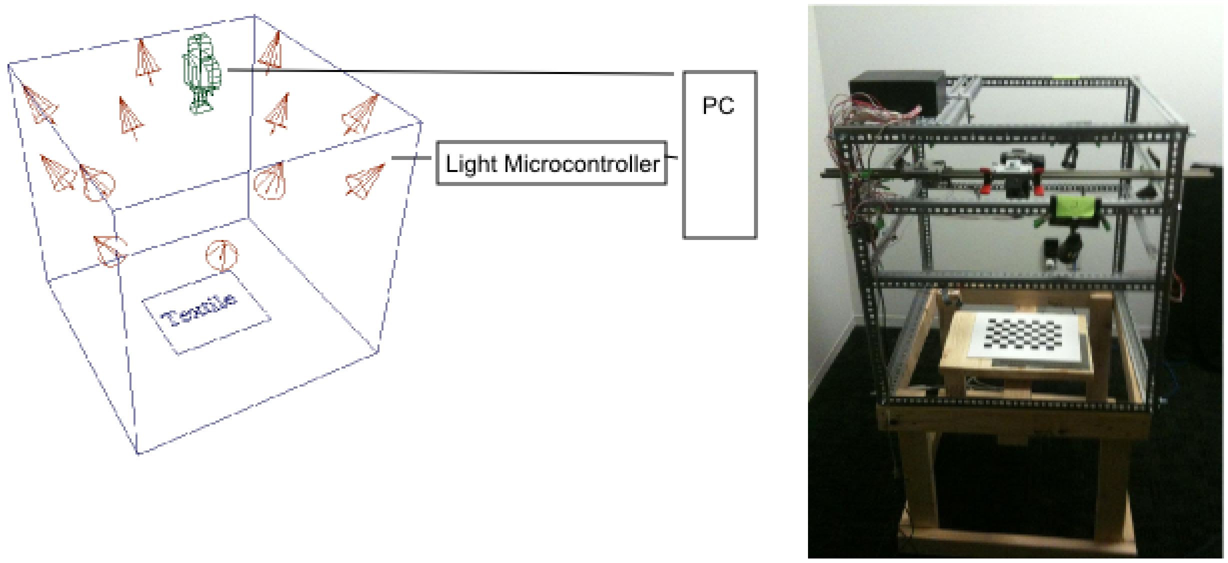


Figure 8: Hardware diagram and actual picture of the hardware

The camera is a Nikon D5100 camera, placed approximately 2 feet above the sample, and hooked to into the computer via a USB cable. The libraw SDK allows us to programmably interface with the camera and control all of its settings, letting us avoid any movement of the camera during operation.

The LEDs are a set of 10 ANSI White (6500k) Rebel LEDs from Luxeon Star. They are evenly distributed in a hemisphere and pointed towards the center of where the fabric sample is placed. The LEDs are controlled by a custom-built light controller, which provides control to individually turn on or off each light. The light controller is an Arduino board coupled with an external power supply, connected to the computer which sends commands to switch between each LED. We've modified the open source program digiCamControl, which gives us a GUI and automates the acquisition of pictures, to also control the light sources.

Camera calibration

We found that calibration of the camera and lights were vital to acquiring good data. Any slight measurement or estimation error propagates into the solver and requires more artist cleanup work to have usable data.

For camera calibration, we needed to solve the camera's position, orientation, and any tangential or radial distortion from the lens. Luckily, using an open source library OpenCV, calibrating a camera for those factors only requires a series of pictures of a checkboard pattern in different positions and orientations to solve for.

Additionally, we needed to correct for non-linear tone-mapping responses to light from the camera. Doing experiments with a book of Matte neutral gray values showed that the Nikon sensor was linear in each channel for most of the dynamic range except at the very extremes of the scale. As long as we took pictures that avoided over-saturating the sensor we can treat the data as linear.

However, either from the inaccuracies from the light or the color filters on the sensor we were not getting a true gray back from the camera. Using the open source library libraw we can obtain the individual values from each sensor in the Bayer pattern and apply our own white balancing based on the measured values in the Matte gray chart. We saved the white balance profile for each light configuration so we could have a consistent white balance across all lighting conditions.

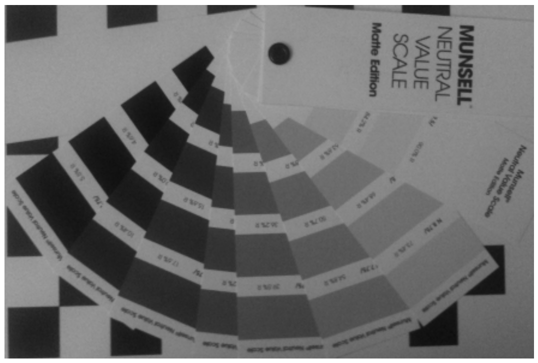


Figure 9: The Munsell Neutral Value Scale Matte chart allowed us to white balance the photos and test the linearity of our sensors.

Light calibration

Initially, we carefully measured the center of the light to the origin. This gave us a rough estimate of position. Using the camera matrix from calibration, we are able to map a pixel position to a world space position on the table. This allows us to get an estimated light direction for each pixel in the image.

Using the Munsell Neutral Value Scale and placing the book flat we are able to solve the equation $(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}}) i k_d = \mathbf{I}$. Since the normal is assumed to be point up $(0, 1, 0)$, the albedo color k_d is ≈ 1, the pixel color I is known from the photo and the light direction is estimated, we can solve for the intensity i for each light: $i = \frac{\mathbf{I}}{l_y}$.

Unfortunately, falloff from the point light, non-uniform angular distribution from the light source, and bad estimates for the initial position of the light will cause gradients towards the edge of the image. The quality will be best near the center of the image where the intensity was computed but gradually get worse towards the edge. For an extreme example please see Figure 10.

To get better intensities and directions we purchased a 99% white diffuse sheet of paper. We flattened the paper onto our table as much as possible and solved for the intensity of each light at each pixel (filtered a bit to remove noise on the paper). This accounts for distance and angular falloff. Using the new intensity we then solve for a new light direction. We can repeat this process a handful of times until we feel that the direction and light intensity map converges.

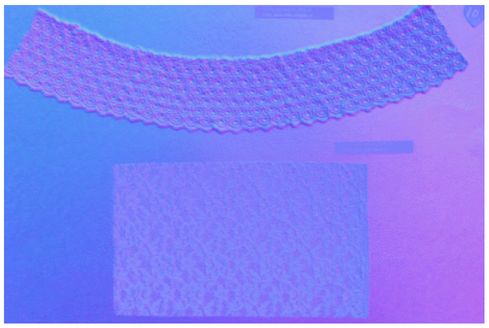


Figure 10: Bad estimation of light intensities, position, direction, falloff (distance/angular) cause low-frequency errors towards the edge of image. Notice the red tendency on the right and cyan on the left. This can be fixed with post processing or better estimations

Normal and Albedo Solver

The scanner takes 10 pictures and from our measurements, we have the known light direction vector parameter $\hat{\mathbf{l}}$ scaled by the measured light intensity i. We have two unknowns that we'd like to solve for, the normal vector $\hat{\mathbf{n}}$ and albedo rgb color k_d.

$$(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})k_d i = I \tag{24}$$

With the assumption of a diffuse material, the equation is the diffuse albedo k_d multiplied by the lighting intensity i and cosine term. First we can extract the light intensity out by dividing both equations by the lighting intensity:

$$(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})k_d = I' \tag{25}$$

We then do four different solves, one for each color channel and one for a grayscale version of the color that we use for calculating the normal:

$$\begin{aligned} \mathbf{I'_r} &= (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})k_{dred} \\ \mathbf{I'_g} &= (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})k_{dgreen} \\ \mathbf{I'_b} &= (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})k_{dblue} \\ \mathbf{I'_l} &= (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})k_{dluminance} \end{aligned} \tag{26}$$

Since we do not know the normal or albedo at the time of the solve, our equation looks more like a scaled version of the normal by the albedo color:

$$\begin{aligned} \mathbf{I'_r} &= \mathbf{N} \cdot \hat{\mathbf{l}} \\ \mathbf{I'_g} &= \mathbf{N} \cdot \hat{\mathbf{l}} \\ \mathbf{I'_b} &= \mathbf{N} \cdot \hat{\mathbf{l}} \\ \mathbf{I'_l} &= \mathbf{N} \cdot \hat{\mathbf{l}} \end{aligned} \tag{27}$$

We have 10 lights so we set up a system of linear equations:

$$\begin{aligned} &I'_{r1} = N_x * l_{x1} + N_y * l_{y1} + N_z * l_{z1} \\ &I'_{r2} = N_x * l_{x2} + N_y * l_{y2} + N_z * l_{z2} \\ &I'_{r3} = N_x * l_{x3} + N_y * l_{y3} + N_z * l_{z3} \\ &\ldots \\ &I'_{r10} = N_x * l_{x10} + N_y * l_{y10} + N_z * l_{z10} \end{aligned} \tag{28}$$

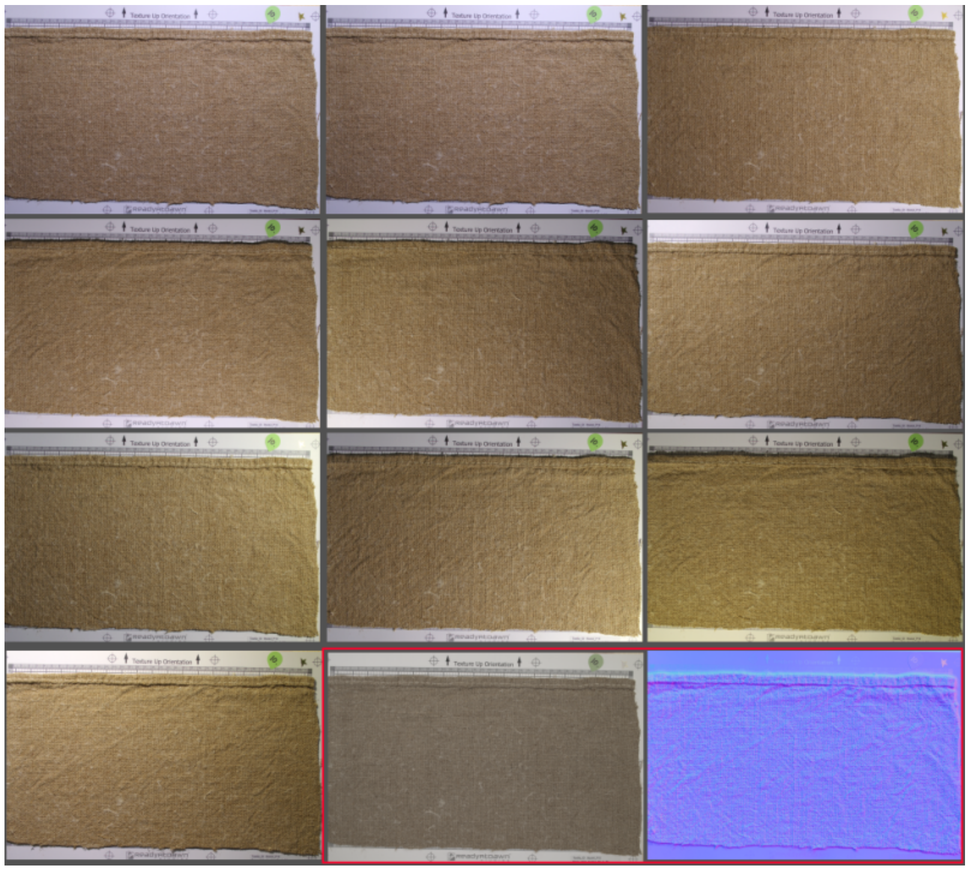


Figure 11: Textile lit from the 10 different lighting directions that is used for solving the normal and albedo. The last two images in the sequence, in the red box, are the solved albedo and normal map. Notice all the detail is captured in the normal map and the albedo is void of any lighting cues.

To solve for the scaled normal N we rearrange terms and use the least squares method to solve. In matrix notation:

$$\begin{aligned} LN &= I \\ L^T LN &= L^T I \\ N &= (L^T L)^{-1} L^T I \end{aligned} \tag{29}$$

In practice, we used the Eigen C++ library and a two-sided Jacobi singular value decomposition. However, the N that is solved for from data has the albedo k_d multiplied into the normalized normal $k_d \hat{N}$, since we know the normal must be normalized, you can extract the albedo by taking the length of the normal $k_d = ||N||$. Then use the length to normalize the normal $\hat{N} = \frac{N}{||N||}$

$$\begin{aligned} k_d &= ||N|| \\ \hat{N} &= \frac{N}{||N||} \end{aligned} \tag{30}$$

Material Authoring Pipeline

Utilizing a physically based shading model was a key component in achieving the level of visual quality that we wanted for our materials. However, it was equally important that we designed a robust and efficient pipeline for authoring individual material assets. Our artists were faced with the challenge of authoring a tremendous number of material variations, while still ensuring that the resulting scene met the performance and memory requirements of our target hardware. To that end, our material authoring pipeline was designed around 3 key components:

- An inheritance-based data format for material assets
- Compositing of multiple materials into parameter maps
- Run-time blending of material layers

Inheritance-based Data Format

One of the core technologies utilized at our studio is an in-house data-description language known as `radattr`. A key feature of this language is that it allows the definition of assets that are derived from a base asset. This allows an asset to share all of the property values defined in the base asset, and selectively *override* any of the properties with new values. Inheritance is a natural fit for materials, because it allows an artist to quickly create a new variation of an existing material by deriving from it and changing one of the parameters. As an example, this is the resulting `radattr` for a simple material featuring several textures and properties, and a second material that derives from the first, in order to create a red version of the material:

```
: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 = (
        :r = 1.0
        :g = 0.0
        :b = 0.0
    )
)
```

The inheritance feature was utilized by our artists to create a standard library of material templates that form the basis of all in-game materials. Material templates represent “pure” versions of core material types such as gold, steel, wood, glass, and concrete. Inheritance is then used to create derived templates of these materials, typically having a different roughness or tint color:

Compositing

Accurately representing the reflectance of real-world objects with a BRDF often requires varying the BRDF parameters across the surface of an object. Typically in real-time applications this is achieved

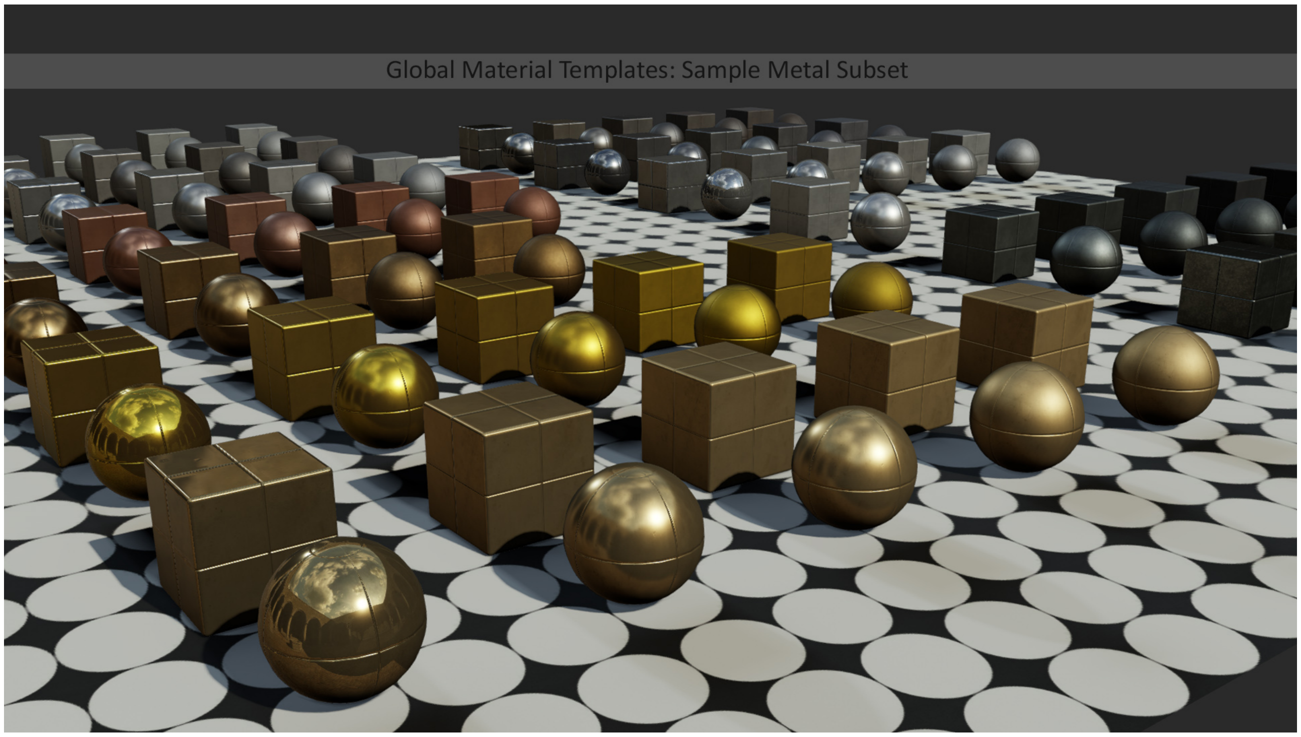


Figure 12: Various material templates used as the base for more complex materials.

by storing BRDF parameters in a 2D texture map that is sampled by the pixel shader, with these maps being hand-authored by texture artists. This approach is extremely inefficient for several reasons:

- Artists are required to associate the desired appearance of a material with colors painted into a texture
- Any transformations applied to material parameters for runtime use, such as range remapping or color space conversions, need to be accounted for when painting the texture
- The properties used for a core material type cannot be changed without changing the contents of all textures that contain the properties of that material type

To avoid these problems, we drew inspiration from offline renderers in designing a compositing system for our material pipeline. In our pipeline, compositing is primarily an offline process that automatically generates texture maps containing material parameters that are ready to be used at runtime by a pixel shader. The parameters placed into these textures are determined by a “stack” of composited materials defined in the material asset, as well as the parameters from the material itself. Our material editor exposes this functionality using a series of composite layers, with a UI resembling the layer system in Photoshop.

In the material editor, the artist can select from a database of existing materials and add one as a new composite layer. The artists can then control certain properties controlling how the layer is composited with other layers, such as an alpha value and a blend mask. When the material is built by our offline content build system, the compositing system evaluates the final runtime value of various parameters for each output texel and then blends it with the next layer in the composite stack. Finally, the composited result is blended with the properties defined in the material itself to determine the final value for all texels in the output texture. To accelerate this process, the blending for each layer is performed on the GPU in a pixel shader.

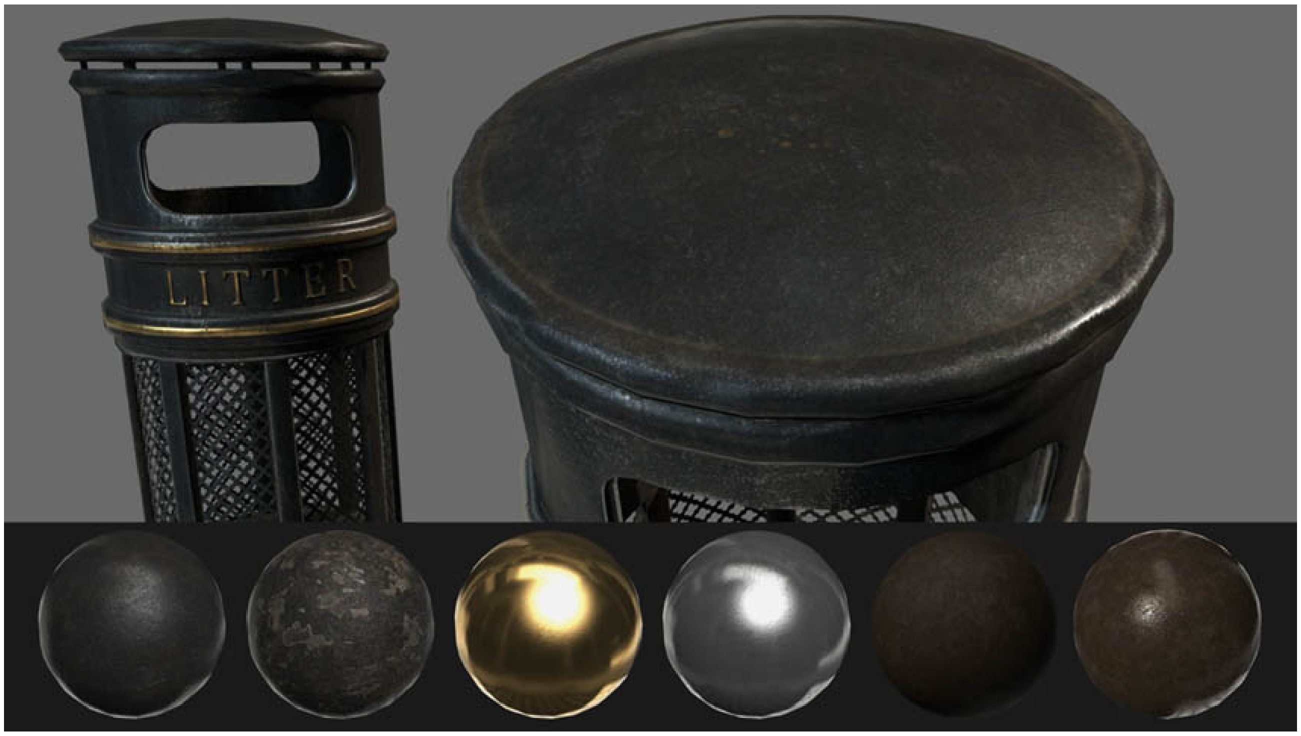


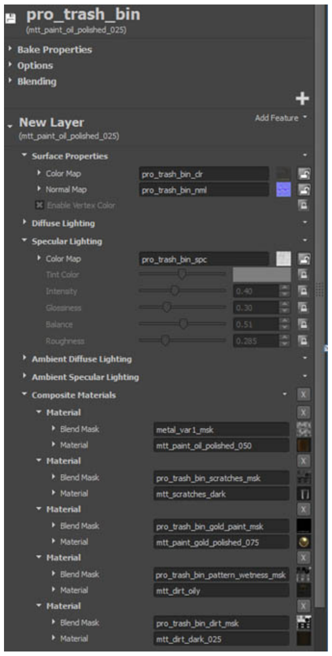
Figure 13: An example of a composited material and the material templates used to generate it.

The materials used as composite layers are typically taken from the material template library. This allows arbitrarily complex surfaces to be formed from a series of simpler, well-understood materials. However, more complex materials, including already-composited materials, can also be used as composite layers. This enables the automated combining of multiple materials into a single material, which helps to reduce draw calls on complex objects. In all cases, the final composited parameter maps are regenerated whenever one of the source materials has changed. This, combined with the inheritance system, supports the propagation of global changes from a single asset. Consequently, we avoid having out-of-date parameters baked into textures that would otherwise have to be manually updated. This level of flexibility has actually allowed us to make changes to our core shading model without our artists having to manually tweak the parameters of every material.

The blending of parameters from composite layers is driven by monochrome blend masks that essentially serve as the alpha channel in the blend equation. These maps can be hand-painted if needed, but can also be automatically generated in certain cases. For instance, a cavity map can be baked from a high-resolution sculpt of a mesh, which can then be used to blend in a composite layer consisting of dirt or grime.

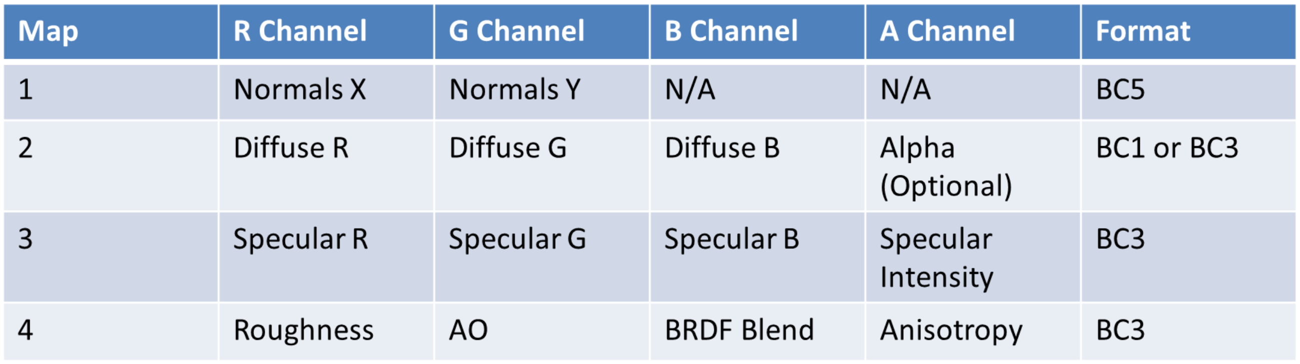
The parameters stored in the textures generated by the compositing pipeline are the inputs required for runtime evaluation of our shading and lighting model. Since the textures are automatically generated, the parameter values can be converted into representations suitable for compact storage and efficient usage in the pixel shader.

To enable a richer level of expressiveness with our materials, we also allow compositing of materials with different BRDFs. This functionality is only available for a subset of our BRDFs, which include our default isotropic GGX BRDF, our anisotropic GGX BRDF, and our cloth BRDF. Compositing of isotropic and anisotropic BRDFs is trivial, since the anisotropic BRDF can be used for both cases. However, compositing of GGX with our cloth BRDF is more involved. For this case we evaluate both BRDFs and blend between the results. This approach is expensive, but allows for complex cases such as water accumulation on top of cloth.



pro_trash_bin
(mtt_paint_oil_polished_025)
Bake Properties
Options
Blending
New Layer
(mtt_paint_oil_polished_025)
Add Feature
Surface Properties
Color Map: pro_trash_bin_dr
Normal Map: pro_trash_bin_nml
Enable Vertex Color
Diffuse Lighting
Specular Lighting
Color Map: pro_trash_bin_spc
Tint Color
Intensity: 0.40
Glossiness: 0.30
Balance: 0.51
Roughness: 0.285
Ambient Diffuse Lighting
Ambient Specular Lighting
Composite Materials
Material
Blend Mask: metal_var1_msk
Material: mtt_paint_oil_polished_050
Material
Blend Mask: pro_trash_bin_scratches_msk
Material: mtt_scratches_dark
Material
Blend Mask: pro_trash_bin_gold_paint_msk
Material: mtt_paint_gold_polished_075
Material
Blend Mask: pro_trash_bin_pattern_wetness_msk
Material: mtt_dirt_oily
Material
Blend Mask: pro_trash_bin_dirt_msk
Material: mtt_dirt_dark_025

Figure 14: The material editor user interface for adding composite layers.



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

Figure 15: Parameter maps output by the compositing pipeline.

Run-Time Layering

To complement the offline compositing system, our material pipeline also features a run-time layering system. The run-time layering has similar functionality to the compositing pipeline, in that it allows parameters from a stack of up to 4 material layers to be blended together based on alpha values. These alpha values typically come from colors embedded in mesh vertices, but can also come from blend maps or even maps dynamically generated at runtime. Typically this functionality is used on environment materials to add unique variation to tiled textures covering a large surface.

In practice, the layering works in a fashion similar to terrain rendering systems. However our inheritance-based asset format allows for additional ease-of-use and efficiency. A layer in a material can be derived from another material asset, which allows artists to quickly select the properties and textures of an existing material for use as a layer in the material they are authoring. Each layer also features its own unique compositing stack, which allows for full flexibility when authoring a layer.

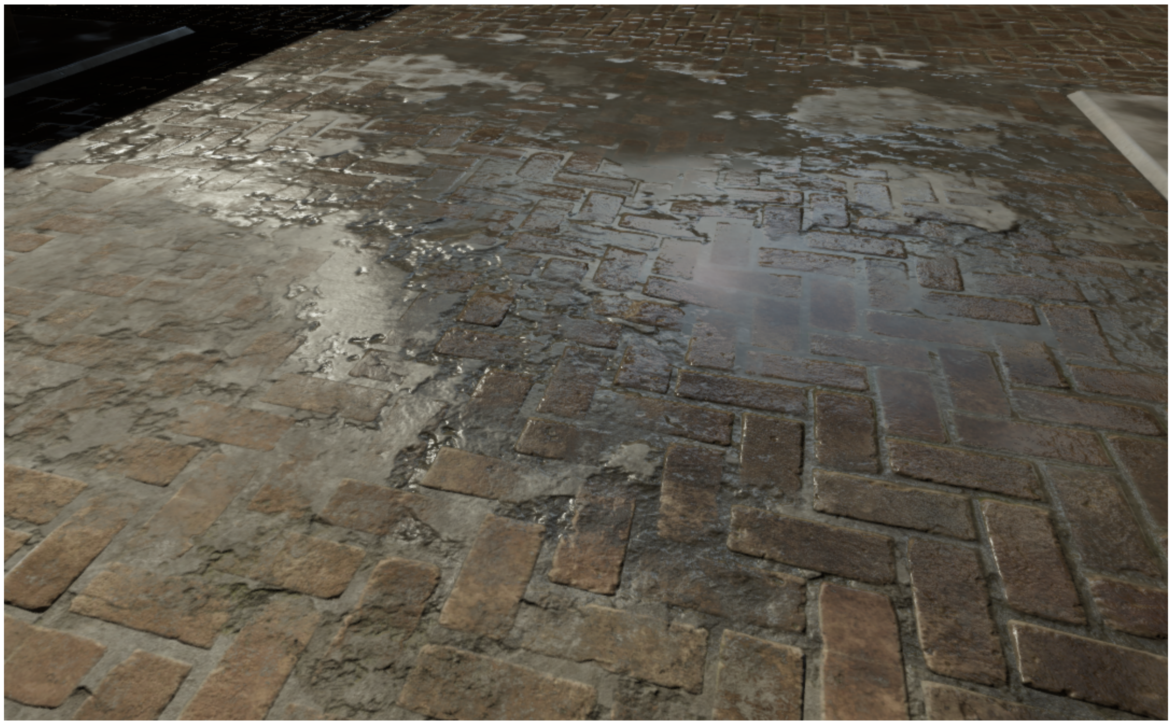


Figure 16: Run-time layering is used to blend mortar and wetness onto a tiling brick material.

Specular Antialiasing

After implementing a physically based shading model for our project, shader aliasing became a common problem for surfaces with low specular roughness. Such aliasing is common in real-time applications due to pixel shader sampling rates being inadequate for the high-frequency reflectance produced by the combination of specular BRDFs and highly detailed normal maps. The issue is compounded by the fact that pixel shader sample locations are fixed in screen-space, which effectively causes them to move over the surface being shaded as the object changes position relative to the camera. The constantly-changing sample positions cause specular highlights to flicker and shimmer over time, which can be highly distracting for the viewer. A more thorough description of this problem can be found in Bruneton and Neyret [2012], as well as an overview of potential solutions.

Frequency Domain Normal Mapping

For our own solution, we initially considered adopting LEAN (Olano and Baker [2010]) or CLEAN (Baker [2011]) mapping. *LEAN Mapping* has been shown to produce excellent results that include anisotropy, however the required precision and additional storage gave us cause for concern. Toksvig's approach from Toksvig [2004] is efficient in terms of memory and computation, but was incompatible with our method for compressing normal maps by discarding the Z component. Techniques for precomputing CLEAN or Toksvig (Hill [2011], McAuley [2012], Baker and Hill [2012]) appeared to be extremely promising, since they could be implemented as a natural extension of our compositing system with no runtime cost. However, we decided to keep searching for an alternative that could give us a generalized framework that we could use with arbitrary BRDFs, as opposed to working with a solution specifically formulated for a single BRDF.

Ultimately, we implemented a solution based on Han et al. [2007b]. The basic concept behind this approach is compute an NDF for each texel of a normal map using all normals from the highest-resolution mip level that contribute to a single lower-resolution texel. This NDF can then be convolved with a BRDF to produce an effective BRDF that properly accounts for the variance of all normal map texels covered by a pixel:

$$f^{\text{eff}}(\mathbf{l},\mathbf{v};\gamma) = \int_{\Omega} f(\mathbf{l},\mathbf{v})\gamma(\mathbf{n})d\mathbf{n} \tag{31}$$

Where $f(\mathbf{l},\mathbf{v})$ is the BRDF, $\gamma(\mathbf{n})$ is the NDF, and $f^{\text{eff}}(\mathbf{l},\mathbf{v};\gamma)$ is the effective BRDF for a given NDF/BRDF combination.

Han utilized spherical harmonics as a framework for performing the convolution in the frequency domain, since convolutions in the spatial domain are equivalent to multiplication in the frequency domain:

$$f^{\text{eff}}_{lm} = \sqrt{\frac{4\pi}{2l+1}} f_l \gamma_{lm} \tag{32}$$

Notice that this convolution takes the same form as the irradiance convolution in equation 18, except that we have substituted the cosine kernel with a generalized BRDF term and the lighting environment with our NDF. We can utilize this equation as long as our BRDF is radially symmetric.

Generation of a spherical harmonics NDF map is trivial. For each texel of the base mip level, a delta oriented with the normal map direction is projected onto the SH basis functions up to a desired order. These SH coefficients can then be linearly filtered in order to compute an NDF for lower resolution mip levels, or to compute the NDF of a pixel at runtime. However, this approach is limited to strictly low-frequency representations of the NDF and BRDF, since high-frequency functions require storing a prohibitive number of SH coefficients.

In order to support high-frequency BRDFs and NDFs, Han proposed storing per-texel NDFs using a spherical radial basis function (SRBF) instead of spherical harmonics. They chose the von Mises-Fisher (vMF) distribution, which essentially provides a normalized Gaussian distribution located on the surface of a unit sphere. Such a distribution is well-suited for a normal distribution function, since a well-behaved NDF should be normalized. A vMF distribution γ with inverse width κ is evaluated for a given direction $\mathbf{n}$ using the cosine of the angle between the direction and the central axis μ:

$$\gamma(\kappa,\mu,\mathbf{n}) = \frac{\kappa}{4\pi\sinh(\kappa)} e^{\kappa(\mathbf{n}\cdot\mu)} \tag{33}$$

To allow vMF lobes to work within the frequency domain framework for NDF convolution, Han developed an analytical formula for approximating the spherical harmonic coefficients of a vMF lobe:

$$\Lambda_l \gamma_l \approx e^{-\frac{l^2}{2\kappa}} \tag{34}$$

Where $\Lambda_l = \sqrt{\frac{4\pi}{2l+1}}$. For convolution with a Torrance-Sparrow BRDF, Han utilized an approximation from Ramamoorthi and Hanrahan [2001]:

$$\hat{f}_l \approx e^{-(\alpha l)^2} \tag{35}$$

Where α is the roughness parameter of the BRDF. Convolving this function with the SH coefficients for our NDF in order to generate an effective BRDF results in the following:

$$\Lambda_l f^{\text{eff}}_l = e^{(\alpha l)^2} e^{-\frac{l^2}{2\kappa}} = e^{(\alpha' l)^2} \tag{36}$$

$$\text{where } \alpha' = \sqrt{\alpha^2 + (2\kappa)^{-1}} \tag{37}$$

This essentially serves as an analytical formula for computing a new, effective roughness for a Torrance-Sparrow BRDF based on the convolution of two Gaussians. In this respect the approach is very similar the solution proposed in Toksvig [2004], which produces a new roughness parameter for a Blinn-Phong BRDF by approximating a Blinn-Phong lobe as a Gaussian.

Implementation in The Order: 1886

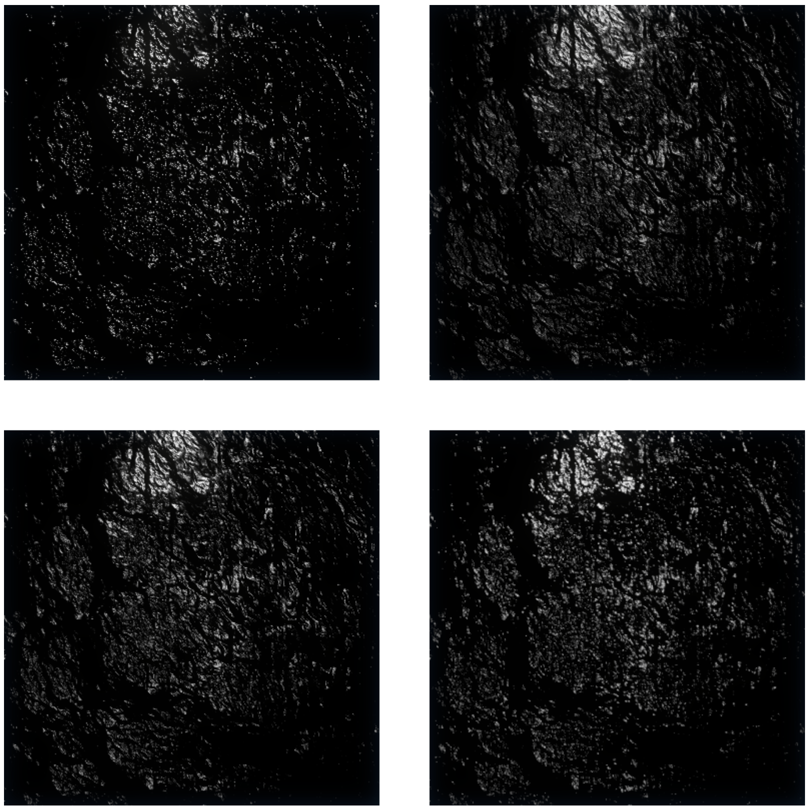


Figure 17: Images demonstrating our solution for antialiasing our specular BRDF. Each image renders a high-frequency normal map under minification, shaded with our GGX specular BRDF with a roughness value of 0.05. Starting with the top-left and going clockwise: no specular antialiasing, pre-computed Toksvig, ground-truth (texture-space lighting), and our solution

To provide specular antialiasing of our materials with no runtime cost, we directly integrated equation 37 into the compositing portion of our material build pipeline. After compositing we have texture maps containing per-texel normal and roughness information, which provides sufficient information for computing an NDF and using it to produce a new roughness value. This new roughness value is stored in the texels of the various mip levels of the roughness map, following the approach used by [McAuley 2012] and [Hill 2012]. The following HLSL code sample demonstrates this process:

```
// ===============================================================================
// Computes a new roughness value for a single mipmap texel given a normal map
// and an existing roughness value
// ===============================================================================
float ComputeRoughness(in float2 texelPos, in uint mipLevel, in float roughness,
   in Texture2D NormalMap)
{
    if(mipLevel == 0)
    {
        return roughness;
    }
    else
    {
        float3 avgNormal = 0.0f;

        // Sample all normal map texels from the base mip level that are within
        // the footprint of the current mipmap texel
        const uint texelFootprint = (1 << mipLevel);
        const float2 topLeft = (-float(texelFootprint) / 2.0f) + 0.5f;
        for(uint y = 0; y < texelFootprint; ++y)
        {
            for(uint x = 0; x < texelFootprint; ++x)
            {
                float2 offset = topLeft + float2(x, y);
                float2 samplePos = floor(texelPos + offset) + 0.5f;
                float3 sampleNormal = NormalMap[samplePos].xyz;
                sampleNormal = normalize(sampleNormal * 2.0f - 1.0f);

                avgNormal += sampleNormal;
            }
        }

        // Fit a vMF lobe to NDF for this mip texel
        avgNormal /= (texelFootprint * texelFootprint);

        float r = length(avgNormal);
        float kappa = 10000.0f;
        if(r < 1.0f)
            kappa = (3 * r - r * r * r) / (1 - r * r);

        // Compute the new roughness value
        return sqrt(roughness * roughness + (1.0f / kappa));
    }
}
```

Ultimately, we were very pleased with the results provided by our solution. As you can see in Figure 17 the amount of aliasing is greatly reduced compared to the image rendered with no specular antialiasing. Temporal stability is also significantly improved, which is a crucial aspect for maintaining high-quality visuals when viewed in motion. In general, the results are very similar to using Toksvig's formula to pre-compute a roughness map, which makes sense given the similarity of the two approaches. In our implementation the Toksvig technique produced slightly higher roughness values in most cases, which caused our vMF-based approach to provide a slightly better match to the ground truth results. Our approach also compares well to LEAN and CLEAN mapping, while avoiding the magnification artifacts inherent to these techniques.

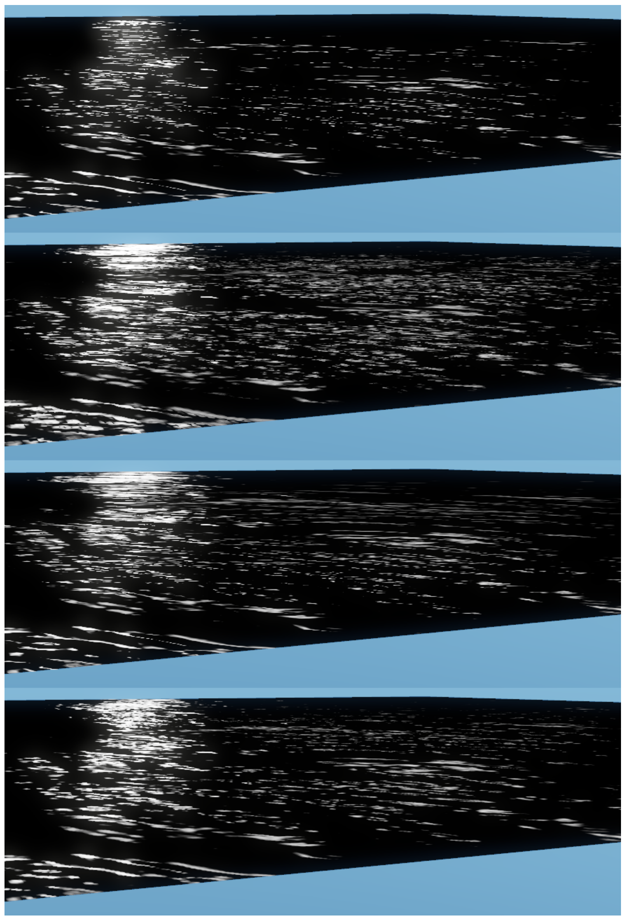


Figure 18: Images showing the results of our specular antialiasing solution compared with LEAN mapping. In these images the surface is being illuminated by a light source placed at a grazing angle, producing highlights in the Fresnel peak. Note that in these images a Beckmann distribution is used, in order to make the results easily comparable with LEAN mapping. Starting with the topmost image and moving downward: no specular antialiasing, LEAN mapping, our solution, and ground truth.

Limitations and Future Work

Our technique for specular antialiasing makes use of several approximations in order to make the roughness pre-computation step efficient. Unfortunately, these approximations are based on assumptions that don't always hold true. The most notable assumption is used in equation 35, which utilizes Ramamoorthi and Hanrahan's approximation of a microfacet BRDF as a simple Gaussian oriented about the reflected light vector. While this assumption generally holds for cases where θ_v is low, it quickly breaks down as θ_v increases and the BRDF enters the Fresnel peak. For these cases the geometry term, Fresnel term, and half-vector parameterization cause large deviations from the simplified model employed by Ramamoorthi and Hanrahan. In fact this limitation is explicitly acknowledged in [Ramamoorthi and Hanrahan 2001], by mentioning that the error of their approximation is only small for small viewing angles and lower SH orders.

In our case, the Gaussian assumption is further violated by our use of a GGX distribution, which has longer tails than the Gaussian-like distributions used in Torrance-Sparrow and Beckmann distributions. Figure 18 shows the results of our antialiasing technique when shading a surface viewed at a grazing angle, demonstrating a case where our approximation is less successful at matching the ground truth.

Another limitation of our technique is due to our use of pre-computed roughness values stored in the

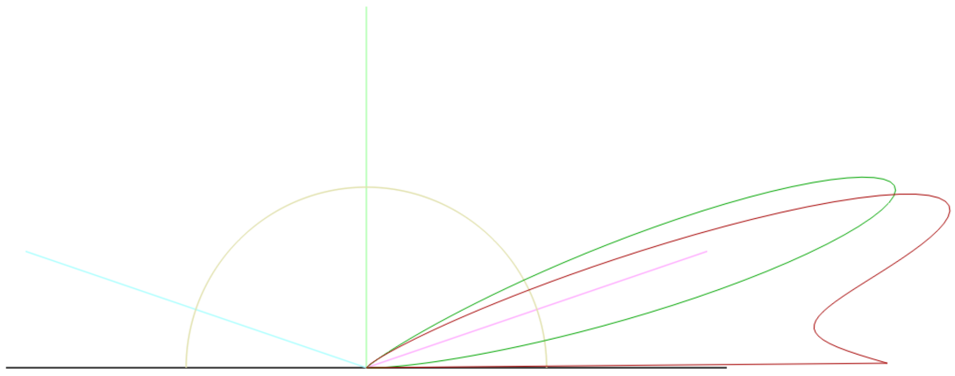


Figure 19: Polar plot of our microfacet BRDF(red) compared with a Phong BRDF(green). The Phong BRDF remains radially symmetrical about the reflected light vector (pink) thus mainting a Gaussian-like shape, while our BRDF significantly deviates from this assumed behavior. Note that that the intensity of the microfacet BRDF has been artificially limited to make the results directly comparable to the Phong BRDF.

mip levels of the roughness map. The grazing angle causes the texture to be sampled at uneven rates with regards to the U and V axis of the texture, which is a situation where anisotropic texture filtering is typically used to prevent aliasing while avoiding over-filtering. In such cases a higher-resolution mip level will be sampled compared to the case where only trilinear filtering is employed. This ultimately prevents specular antialiasing from occuring when anisotropic filtering is used to sample the roughness texture, since the higher-resolution mip level will have only considered a small neighborhood of normal map texels. To remedy this issue we use trilinear filtering when sampling the roughness map, which allows the higher roughness values to take effect at grazing angles. However this causes the opposite problem of over-estimating the roughness, due to considering normal map texels outside of the pixel footprint. See Figure 20 for an example of this problem. In such cases a technique such as LEAN mapping has the advantage, since LEAN mapping computes normal variance on-the-fly based on filtered normal map results. In practice we've found that over-estimating the roughness map produces higher-quality results than using anisotropic filtering due to the temporal artifacts that can occur due to the increased aliasing when sampling a higher-resolution mip level.

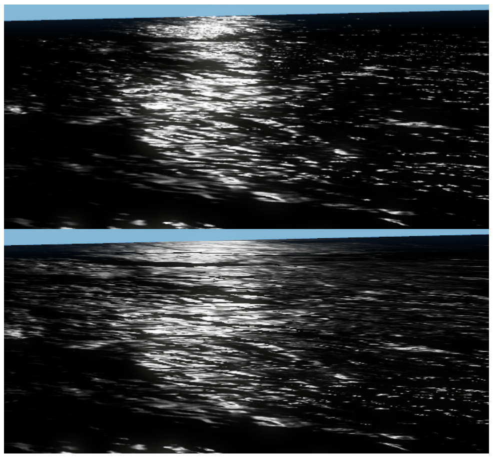


Figure 20: Images showing the results of roughness over-estimation due to trilinear texture filtering of the roughness map. The top image is ground truth, and the bottom image is our solution with trilinear filtering used to sample the roughness map.

Despite these limitations, we've found that our technique provides adequate results in terms of preventing unwanted specular aliasing artifacts in our game. We see this technique as a starting point for future improvments that utilize the generalized frequency-domain framework to produce approximations that provide a better fit for our shading models. In the future we hope to account for the actual behavior of microfacet BRDF's at grazing angles, as well as develop formulations specific to the GGX distribution. Another potential area of research is solving for anisotropic parameters based on the NDF, in order to approximate the results achieved in the original paper by solving for multiple vMF lobes. Antialiasing of the diffuse term based on the NDF is also something we've considered, however we have not yet implemented this in our engine.

Acknowledgments

We would like to thank the programmers Aaron Halstead for helping develop the material scanner, Garret Foster for initial development of the material compositing pipeline, Joe Schutte for initial development of the Cloth BRDF. Also, we'd like to thank our artists Nathan Phail-Liff and Anthony Vitale for pushing us to make great features. In addition, we'd like to give an extra-special thanks to Stephen Hill and Stephen McAuley for their assistance in preparing our course material, as well as for suggesting the visibility term optimization for GGX.

Appendix A: HLSL Shader Code for GGX and Beckmann Specular BRDFs

```
// ================================================================================
// Calculates the Fresnel factor using Schlick's approximation
// ================================================================================
float3 Fresnel(in float3 specAlbedo, in float3 h, in float3 l)
{
    float lDotH = saturate(dot(l, h));
    return specAlbedo + (1.0f - specAlbedo) * pow((1.0f - lDotH), 5.0f);
}

// ================================================================================
// 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, with a matching
// geometry factor and visibility term. m is roughness, n is the surface normal,
// h is the half vector, l is the direction to the light source, and specAlbedo is
// the RGB specular albedo
// ================================================================================
float3 GGX_Specular(in float m, in float3 n, in float3 h, in float3 v, in float3 l,
                    in float3 specAlbedo)
{
    float nDotL = saturate(dot(n, l));
    if(nDotL <= 0.0f)
        return 0.0f;

    float nDotH = saturate(dot(n, h));
    float nDotV = max(dot(n, v), 0.0001f);

    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;

    // Calculate the fresnel term
    float f = Fresnel(specAlbedo, h, l);

    // Put it all together
    return d * f * vis;
}

// ===============================================================================
// Helper for computing the Beckmann geometry term
// ===============================================================================
float Beckmann_G1(float m, float nDotX)
{
    float nDotX2 = nDotX * nDotX;
    float tanTheta = sqrt((1 - nDotX2) / nDotX2);
    float a = 1.0f / (m * tanTheta);
    float a2 = a * a;

    float g = 1.0f;
    if(a < 1.6f)
        g *= (3.535f * a + 2.181f * a2) / (1.0f + 2.276f * a + 2.577f * a2);

    return g;
}

// ===============================================================================
// Computes the specular term using a Beckmann microfacet distribution, with a
// matching geometry factor and visibility term. m is roughness, n is the surface
// normal, h is the half vector, l is the direction to the light source, and
// specAlbedo is the RGB specular albedo
// ===============================================================================
float3 Beckmann_Specular(in float m, in float3 n, in float3 h, in float3 v,
                         in float3 l, in float3 specAlbedo)
{
    float nDotL = saturate(dot(n, l));
    if(nDotL <= 0.0f)
        return 0.0f;

    float nDotH = saturate(dot(n, h));
    float nDotV = max(dot(n, v), 0.0001f);
    float nDotH2 = nDotH * nDotH;
    float nDotH4 = nDotH2 * nDotH2;
    float m2 = m * m;

    // Calculate the distribution term
    float tanTheta2 = (1 - nDotH2) / nDotH2;
    float expTerm = exp(-tanTheta2 / m2);
    float d = expTerm / (Pi * m2 * nDotH4);

    // Calculate the matching geometric term
    float g1i = Beckmann_G1(m, nDotL);
    float g1o = Beckmann_G1(m, nDotV);
    float g = g1i * g1o;

    // Calculate the fresnel term
    float f = Fresnel(specAlbedo, h, l);

    // Put it all together
    return d * g * f * (1.0f / (4.0f * nDotL * nDotV));
}
```

Bibliography

Michael Ashikhmin and Simon Premoze. Distribution-based brdfs. *Unpublished Technical Report, University of Utah*, 2, 2007.

Dan Baker. Spectacular specular - lean and clean specular highlights. GDC 2011, 2011.

Dan Baker and Stephen Hill. Rock-solid shading - image stability without sacrificing detail. SIGGRAPH 2012 Course: Practical Physically Based Shading in Film and Game Production, 2012.

E. Bruneton and F. Neyret. A survey of nonlinear prefiltering methods for efficient and accurate surface shading. *Visualization and Computer Graphics, IEEE Transactions on*, 18(2):242–260, 2012. ISSN 1077-2626. doi: 10.1109/TVCG.2011.81.

Brent Burley. Physically-based shading at disney. *part of Practical Physically-Based Shading in Film and Game Production, SIGGRAPH*, 2012.

R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1 (1):7–24, January 1982. ISSN 0730-0301. doi: 10.1145/357290.357293. URL `http://doi.acm.org/10.1145/357290.357293`.

Charles Han, Bo Sun, Ravi Ramamoorthi, and Eitan Grinspun. Frequency domain normal map filtering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):28:1–28:12, 2007a.

Charles Han, Bo Sun, Ravi Ramamoorthi, and Eitan Grinspun. Frequency domain normal map filtering. In *ACM Transactions on Graphics (TOG)*, volume 26, page 28. ACM, 2007b.

Stephen Hill. Specular showdown in the wild west. 2011. URL `http://blog.selfshadow.com/2011/07/22/specular-showdown/`.

Dimitar Lazarov. Physically based lighting in call of duty: Black ops. SIGGRAPH 2011 Course: Advances in Real-Time Rendering in 3D Graphics and Games, 2011.

Stephen McAuley. Calibrating lighting and materials in far cry 3. SIGGRAPH 2012 Course: Practical Physically Based Shading in Film and Game Production, 2012.

Marc Olano and Dan Baker. Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 181–188. ACM, 2010.

Eric Penner and George Borshukov. Pre-integrated skin shading. *Gpu Pro 2*, 2:41, 2011.

Ramamoorthi, P. Hanrahan, Ravi Ramamoorthi, and Pat Hanrahan. On the relationship between radiance and irradiance: determining the illumination from images of a convex lambertian object. 2001.

Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 117–128, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383271. URL `http://doi.acm.org/10.1145/383259.383271`.

Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246. Wiley Online Library, 1994.

P. Shirley, B. Smits, H. Hu, and E. Lafortune. A practitioners' assessment of light reflection models. In *Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, PG '97, pages 40–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8028-8. URL `http://dl.acm.org/citation.cfm?id=826026.826423`.

Peter S. Shirley. *Physically based lighting calculations for computer graphics*. PhD thesis, Champaign, IL, USA, 1991. UMI Order NO. GAX91-24487.

Michael Toksvig. Mipmapping normal maps. Technical report, 2004.

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, EGSR'07, pages 195–206, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. ISBN 978-3-905673-52-4. doi: 10.2312/EGWR/EGSR07/195-206. URL `http://dx.doi.org/10.2312/EGWR/EGSR07/195-206`.

Robert J Woodham. Photometric method for determining surface orientation from multiple images. *Optical engineering*, 19(1):191139–191139, 1980.