Samurai Shading

in

Jasmin Patry
Sucker Punch Productions

Speaker notes

Hi! My name is Jasmin Patry, and I work at Sucker Punch Productions as a lead rendering engineer. I'm honored and excited to be talking to you about some of the physically based shading research that we performed while working on our new PlayStation 4 game, Ghost of Tsushima.

Introduction

GHOST

- Sucker Punch is a part of Sony Interactive Entertainment Worldwide Studios (soon PlayStation Studios)
- Peak size: 160 people (including 25 QA)
- Previous games include:
  - *Sly Cooper* (*1*, *2*, & *3*)
  - *Infamous* (*1*, *2*, *Festival of Blood*, *Second Son*, *First Light*)

Speaker notes

If you're not familiar with Sucker Punch, we've been a part of Sony Interactive Entertainment since 2011, and are located in Bellevue, Washington, near Seattle.
[next]
During the development of Ghost, we reached a peak of 160 people working in our studio,
[next]
and our previous games include
[next]
the Sly Cooper series,
[next]
and the Infamous series of games.

Introduction

- Recently released *Ghost of Tsushima* for the PlayStation 4

Speaker notes

As I mentioned, we've recently released Ghost of Tsushima for the PlayStation 4.

Speaker notes

Ghost is an open-world action adventure set in 13th-century feudal Japan...

Speaker notes

... in which you take on the role of samurai Jin Sakai...

Speaker notes

... the lone survivor of the Mongol invasion of the island of Tsushima...

Speaker notes

... as he fights to liberate his home.

Introduction | Goals

- Our goals after *Infamous: Second Son* and *First Light* included improving the quality of our materials and lighting
  - Particularly for characters
- **Goal:** Support new types of materials, including:
  - Materials with strong anisotropic specular response
  - Materials with strong asperity scattering component
- **Goal:** Improve fidelity of skin shading
- **Goal:** Make it easy to use detail maps based on scanned materials
  - For all types of maps (albedo, gloss, specular, normal, etc.)

Speaker notes

After finishing Infamous: Second Son and First Light, one of our goals was to improve the quality of our materials and lighting,
[next]
particularly for our character.
[next]
One goal was to support new types of materials, such as silk, which has a strong anisotropic specular response, as well as materials like velvet, with strong asperity scattering.
[next]
We also wanted to improve the fidelity of our skin shading; in particular the quality of our subsurface scattering.
[next]
Another goal was to make it easy to use detail maps based on scanned materials;
[next]
We'd already been using normal detail maps, but we wanted to make it easy and intuitive to author materials with albedo, gloss, and specular detail maps, in order to keep our texture sizes down, while retaining detail close up.

Introduction | Lighting

- Art direction called for “stylized realism”
- Lighting models are physically based
- Materials authored with physically plausible values, some photogrammetry
- Lighting can deviate from physical correctness
  - Artists can globally adjust to achieve desired look

Speaker notes

Now I'll talk a little bit about the lighting on Ghost; our art direction called for "stylized realism" -- so we weren't aiming for photorealism on this project.
[next]
Our lighting models are physically based, however, and our materials are authored using physically plausible values, with some photogrammetry. This allowed our materials to respond to light in a consistent way.
[next]
Where we do deviate from physical correctness is at the lighting stage; we provide a set of knobs to the lighting artists that allow them to globally adjust various lighting parameters (such as ambient diffuse and specular balance, sky brightness, etc.) to achieve the desired look. Typically this is most commonly done at the granularity of the weather and time of day states in our weather system.

[Additional Notes]
Other physically based lighting bits:
* Energy conservation
* GGX specular NDF
* Smith visibility function
* Schlick Fresnel
* Lambertian diffuse + optional energy-conserving translucency + optional fuzziness
* SH derived from sky visibility convolved with sky SH
* Dynamically relit ambient specular probes, convolved with GGX NDF at runtime
* Physical lighting values
* Energy conserving area lights

- Most lighting in *Ghost* is deferred:
  - Lambertian diffuse
    - Optional translucency
    - Optional simplified asperity scattering (fuzziness)
  - Isotropic GGX specular
    - Optional colored specular via “metallic” channel
- Other materials are forward lit, e.g. when using:
  - Anisotropic GGX specular
  - Full anisotropic asperity scattering BRDF
  - Subsurface scattering

Speaker notes

In terms of the rendering technology, most lighting in Ghost is deferred;
[next]
By default we use Lambertian diffuse,
[next]
with an optional translucent lighting model,
[next]
as well as an optional asperity scattering model.
[next]
Specular uses isotropic GGX,
[next]
with optional colored specular via a metallic channel.
[next]
Materials with different requirements are forward-lit, for example when using
[next]
anisotropic GGX specular,
[next]
the full anisotropic asperity scattering BRDF that I'll be discussing later in this talk,
[next]
or subsurface scattering.
[next]
These four highlighted points are ones that I'll be discussing in more detail.

Introduction | Overview

GHOST

- This talk is divided into the following four topics:
  - Anisotropic specular maps and filtering
  - Anisotropic asperity scattering BRDF
  - Skin shading improvements
  - Physically based detail maps (as bonus slides)




Speaker notes

I've divided this talk into four sections:
[next]
First I'll discuss our approach to storing and filtering anisotropic specular maps;
[next]
Next I'll discuss our new anisotropic asperity scattering BRDF;
[next]
I'll then discuss the improvements that we made to our subsurface scattering model;
[next]
The final section discussing detail maps will be available as bonus slides only, due to time constraints.

Speaker notes

So to begin, let me describe our approach to storing and filtering anisotropic specular maps.

Anisotropic Specular

- In previous games, we supported GGX anisotropic specular aligned with mesh tangent space
- Wanted to support arbitrary spatially varying orientation
  - Plus normal variance filtering
- The SGGX paper by Heitz et al. (2015) gave us ideas on ways to accomplish both goals

Speaker notes

In our earlier games, we supported anisotropic GGX specular, but it was restricted to the alignment of the mesh's tangent space.
[next]
For Ghost we wanted to support arbitrary, spatially varying orientation, partially because we were evaluating scanned materials which provided this data, and we wanted to be able to visualize it in engine.
[next]
We also wanted to support normal variance filtering, like we already did for our isotropic specular normal and gloss maps.
[next]
Around the same time that we were working on this, the SGGX paper by Heitz and others was published, which gave us ideas for new ways that we might achieve these goals.

- LEAN Mapping (Olano and Baker, 2010)
  - Uses a Beckmann-like NDF
  - Requires choice of scale factor for texture storage
    - 16-bit textures preferable
  - For flat normals, gives equivalent results to our algorithm
  - Can be adapted to work with GGX-based BRDFs using:

$$\sigma = \frac{1}{\sqrt{2}}\alpha$$

    - σ: Beckmann roughness
    - This is the approach that we used in the following comparisons

Speaker notes

There has been previous work done in this area, and the first that I'll discuss is LEAN Mapping from 2010.
[next]
It uses a Beckmann-like NDF to filter normal maps to produce an anisotropic antialiased result.
For storage, it requires the careful choice of a scaling factor, especially when using 8-bit textures, which makes 16-bit storage preferable.
[next]
For flat normals, LEAN filtering gives results that our equivalent to our algorithm, but the results do deviate as normals get steeper.
[next]
While LEAN filtering is based on the Beckmann NDF, it can be adapted to work with GGX-based BRDFs using this equation, where sigma is the Beckmann roughness from the formulation used by the paper, while alpha is the GGX roughness. This is what we've done for the following comparisons.

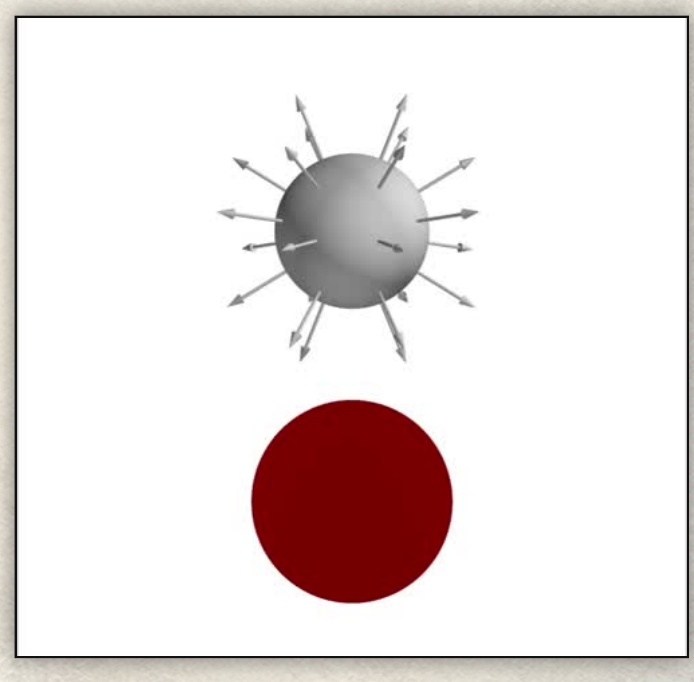
- LEADR Mapping (Dupuy, Heitz, et al., 2013)
  - Incorporates displacement mapping
  - Accounts for entire BRDF:
    - NDF
    - Masking-shadowing
    - Diffuse microfacets
  - Uses Beckmann NDF, and same storage and NDF filtering as LEAN
  - Similar to our approach, except that we're:
    - Using the GGX NDF and associated shadow-masking term
    - Using SGGX-based filtering instead of Beckmann-based
    - Not including diffuse microfacets or displacement mapping

Speaker notes

LEADR mapping from 2013 builds on LEAN mapping,
[next]
by incorporating displacement mapping and accounting for the entire BRDF, and not just the normal distribution function.
[next]
It also uses the Beckmann NDF, and the same storage and NDF filtering approach as LEAN.
[next]
LEADR is similar to our approach, except that we're using the GGX NDF and associated Smith shadow-masking term, and we're using a new SGGX-based filtering method that I'll describe shortly. We're also not including diffuse microfacets or displacement mapping here.

Anisotropic Specular | SGGX Recap

- SGGX developed as a *microflake distribution*
  - Extends GGX NDF to spherical domain
  - NDF of ellipsoid
  - GGX is a special case of SGGX



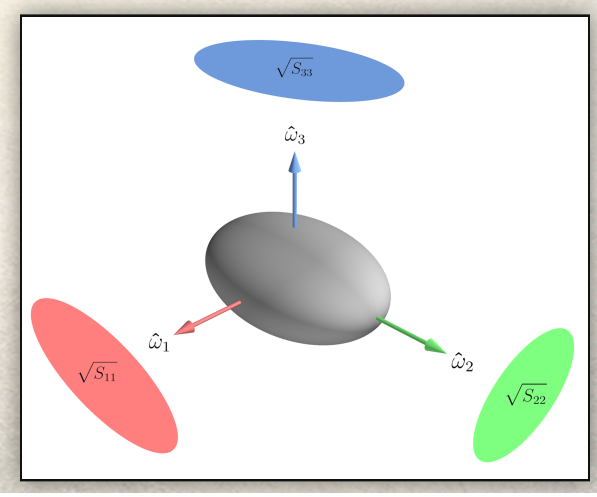
Speaker notes

I'll give a brief over of SGGX; it was developed as a microflake distribution for representing anisotropic participating media.
[next]
It extends the GGX normal distribution function from the hemispherical domain to the spherical domain.
It describes the NDF of an ellipsoid; if you were to plot the density of the normals of the animating ellipsoid on the right, you would get the heat map shown below the ellipsoid.
[next]
Another nice property of SGGX is that GGX is a special case, which makes it possible to leverage it for use for our application, which is surface shading.

- SGGX characterized by an $\mathbf{S}$ matrix

$$\mathbf{S} = [\hat{\omega}_1, \hat{\omega}_2, \hat{\omega}_3] \begin{bmatrix} S_{11} & 0 & 0 \\ 0 & S_{22} & 0 \\ 0 & 0 & S_{33} \end{bmatrix} [\hat{\omega}_1, \hat{\omega}_2, \hat{\omega}_3]^T$$

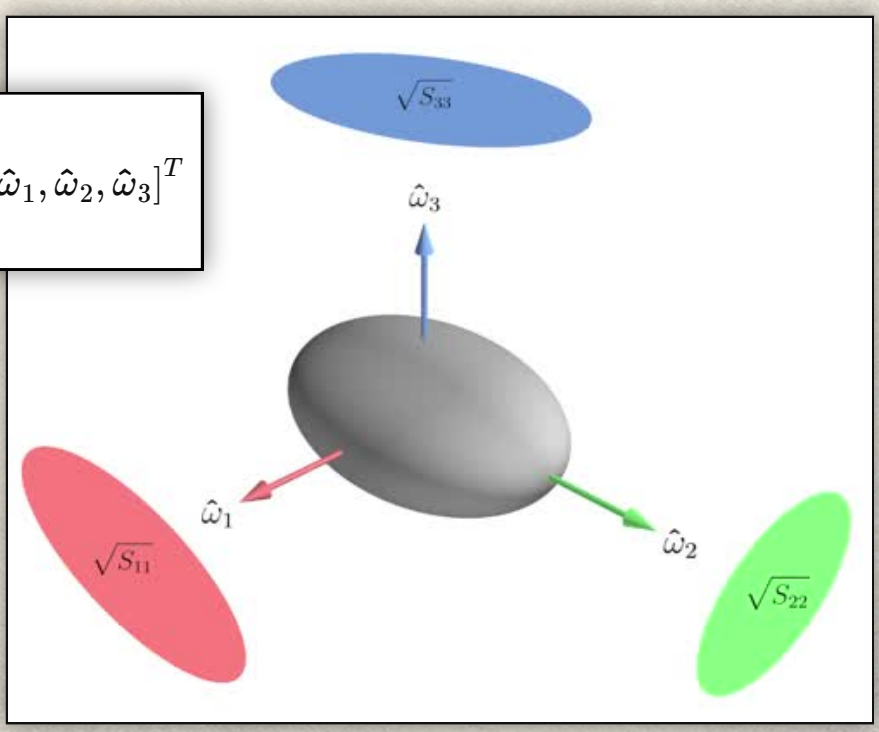
- $\hat{\omega}_i$: principal axes of ellipsoid
  - Also eigenvectors of $\mathbf{S}$
- S_{ii}: Square of ellipsoid projected area
  - Eigenvalues of $\mathbf{S}$



Speaker notes

SGGX is characterized by an S matrix,
[next]
which is defined like so
[next]
where the omega i's are the principal axes of the ellipsoid (and also the eigenvectors of S)
[next]
while the terms on the diagonal are eigenvalues, and also the squares of the ellipsoid projected areas on each axis...

$$\mathbf{S} = [\hat{\omega}_1, \hat{\omega}_2, \hat{\omega}_3] \begin{bmatrix} S_{11} & 0 & 0 \\ 0 & S_{22} & 0 \\ 0 & 0 & S_{33} \end{bmatrix} [\hat{\omega}_1, \hat{\omega}_2, \hat{\omega}_3]^T$$



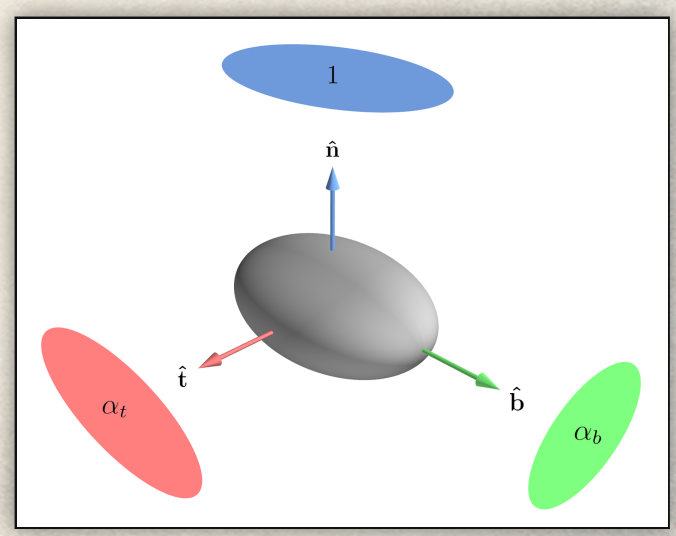
Speaker notes

...which can be better seen here.

- GGX is a special case:

$$\mathbf{S} = \left[\hat{\mathbf{t}}, \hat{\mathbf{b}}, \hat{\mathbf{n}}\right] \begin{bmatrix} \alpha_t^2 & 0 & 0 \\ 0 & \alpha_b^2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left[\hat{\mathbf{t}}, \hat{\mathbf{b}}, \hat{\mathbf{n}}\right]^T$$

- $\hat{\mathbf{t}}$: Tangent
- $\hat{\mathbf{b}}$: Bitangent
- $\hat{\mathbf{n}}$: Normal
- α: GGX roughness



Speaker notes

As I mentioned, GGX is a special case,
[next]
and can be written like so
[next]
where the projected area in the direction of the normal is defined to be 1, while the projected area in the direction of the tangent is alpha_t, and the projected area in the direction of the bitangent is alpha_b. It's important to note that GGX is only defined on the upper half of this ellipsoid.

- Using the normal-mapped basis defined by $\hat{\mathbf{t}}_n$, $\hat{\mathbf{b}}_n$, and $\hat{\mathbf{n}}$, GGX can also be written as:

$$\mathbf{S} = \left[\hat{\mathbf{t}}_n, \hat{\mathbf{b}}_n, \hat{\mathbf{n}}\right] \begin{bmatrix} S_{xx} & S_{xy} & 0 \\ S_{xy} & S_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \left[\hat{\mathbf{t}}_n, \hat{\mathbf{b}}_n, \hat{\mathbf{n}}\right]^T$$

- $\hat{\mathbf{t}}_n$: Normal-mapped tangent
- $\hat{\mathbf{b}}_n$: Normal-mapped bitangent
- $\hat{\mathbf{n}}$: Normal-mapped normal
- S_{xx}, S_{xy}, S_{yy}: 2x2 GGX submatrix

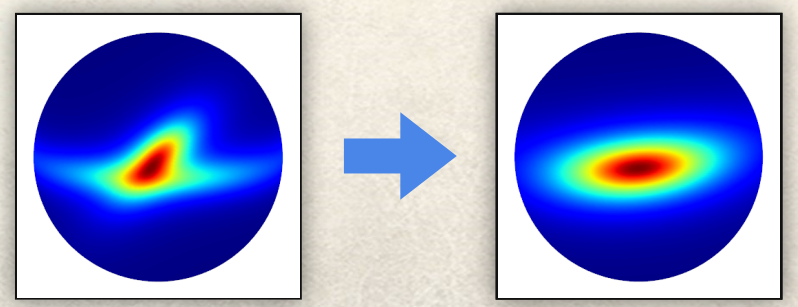
Speaker notes

We can also write GGX
[next]
in this form, using the mesh's tangent space after normal mapping (instead of the anisotropic tangent and bitangent vectors), with the rotation of the anisotropic basis incorporated in to the 2x2 upper left submatrix. This is more compact in terms of storage, because we only need to store the three submatrix terms.
[next]
t_n and b_n are the normal-mapped tangent and bitangent,
[next]
while n is the normal-mapped normal, which together form an orthonormal basis.
[next]
These three terms form the 2x2 symmetric GGX submatrix, which encodes both the anisotropic roughness and orientation.

[Additional Notes]
This slide is corrected from the original version which incorrectly used unit vectors i and j instead of t_n and b_n.

Anisotropic Specular | SGGX Recap

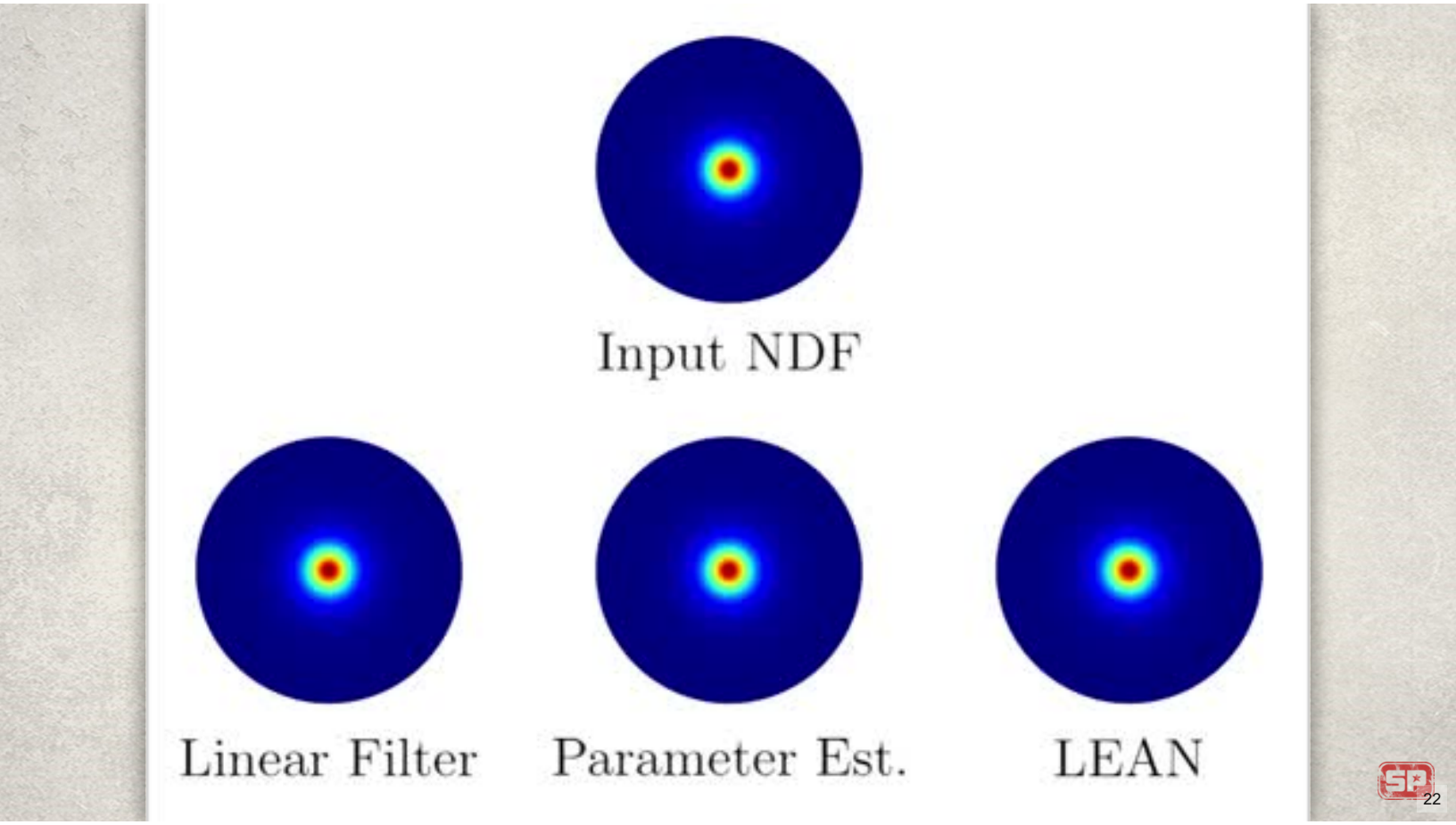
- The SGGX paper describes a “parameter estimation” algorithm for determining the SGGX NDF from an arbitrary NDF
  - SGGX NDF ellipsoid has same axes as covariance eigenvectors
  - Preserves microflake projected area along these axes
- Heitz et al. also show that linearly filtering $\mathbf{S}$ is a good approximation to this algorithm



Input NDF Linear Filtering

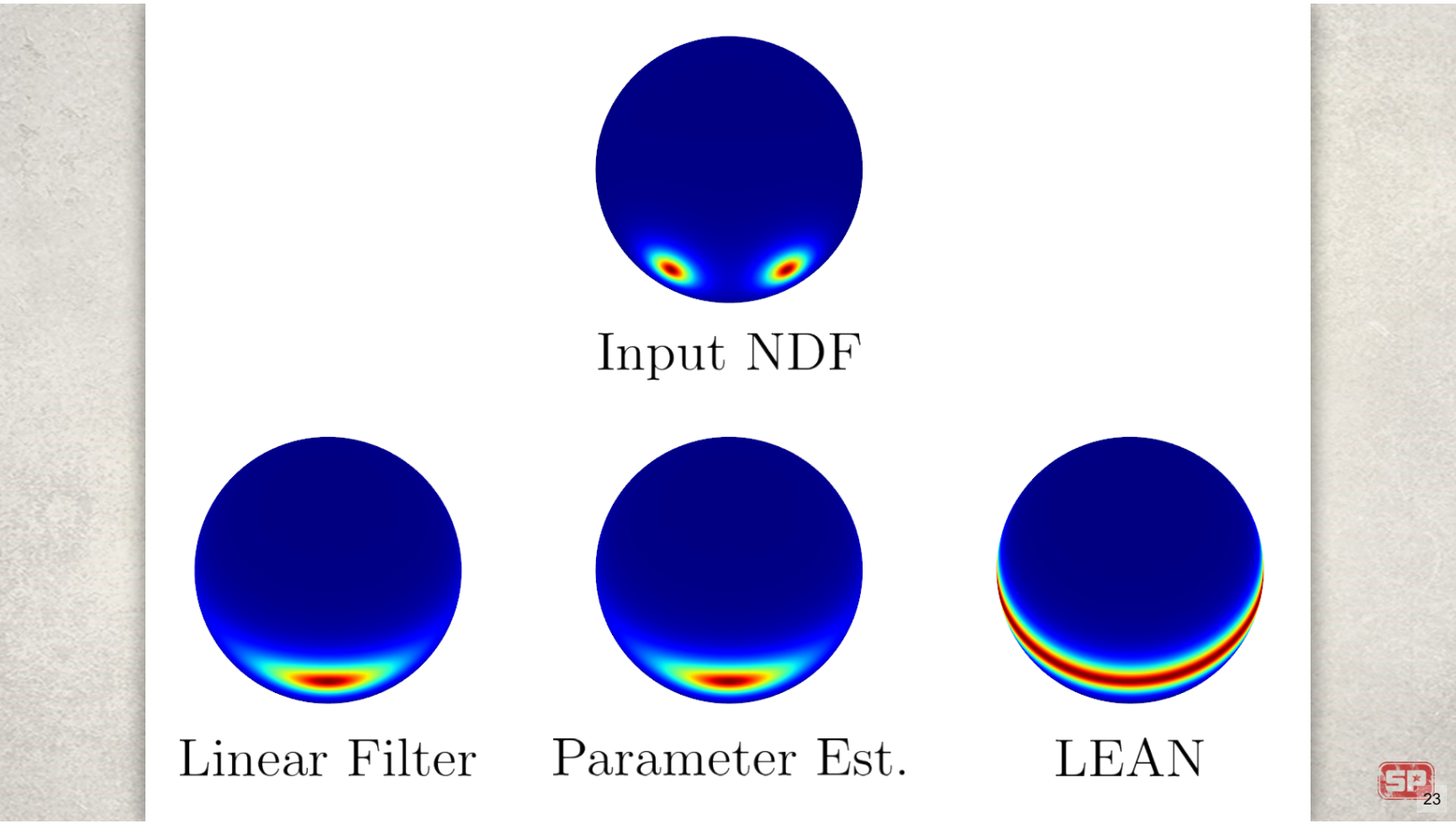
Speaker notes

The SGGX paper describes a parameter estimation algorithm, which takes as input an arbitrary NDF and produces an SGGX NDF with the following properties:
[next]
The ellipsoid has the same axes as the eigenvectors of the input NDF's covariance matrix
[next]
and it preserves the microflake projected area along these axes. This means that if the algorithm is given an SGGX NDF as input, it will return the same NDF. These seemed to us like desirable properties for a GGX filtering algorithm to have.
[next]
As an example, here is an input NDF which is a linear combination of two SGGX NDFs.
[next]
Applying the parameter estimation algorithm to it results in the NDF on the right.
[next]
The paper shows that if the input NDF is a mixture of SGGX NDFs, then filtering their S matrices usually yields a good approximation to parameter estimation.
[next]
The result of linear filtering looks like this; as you can see it's quite close to the result of parameter estimation. In fact, for many cases the match is virtually exact.



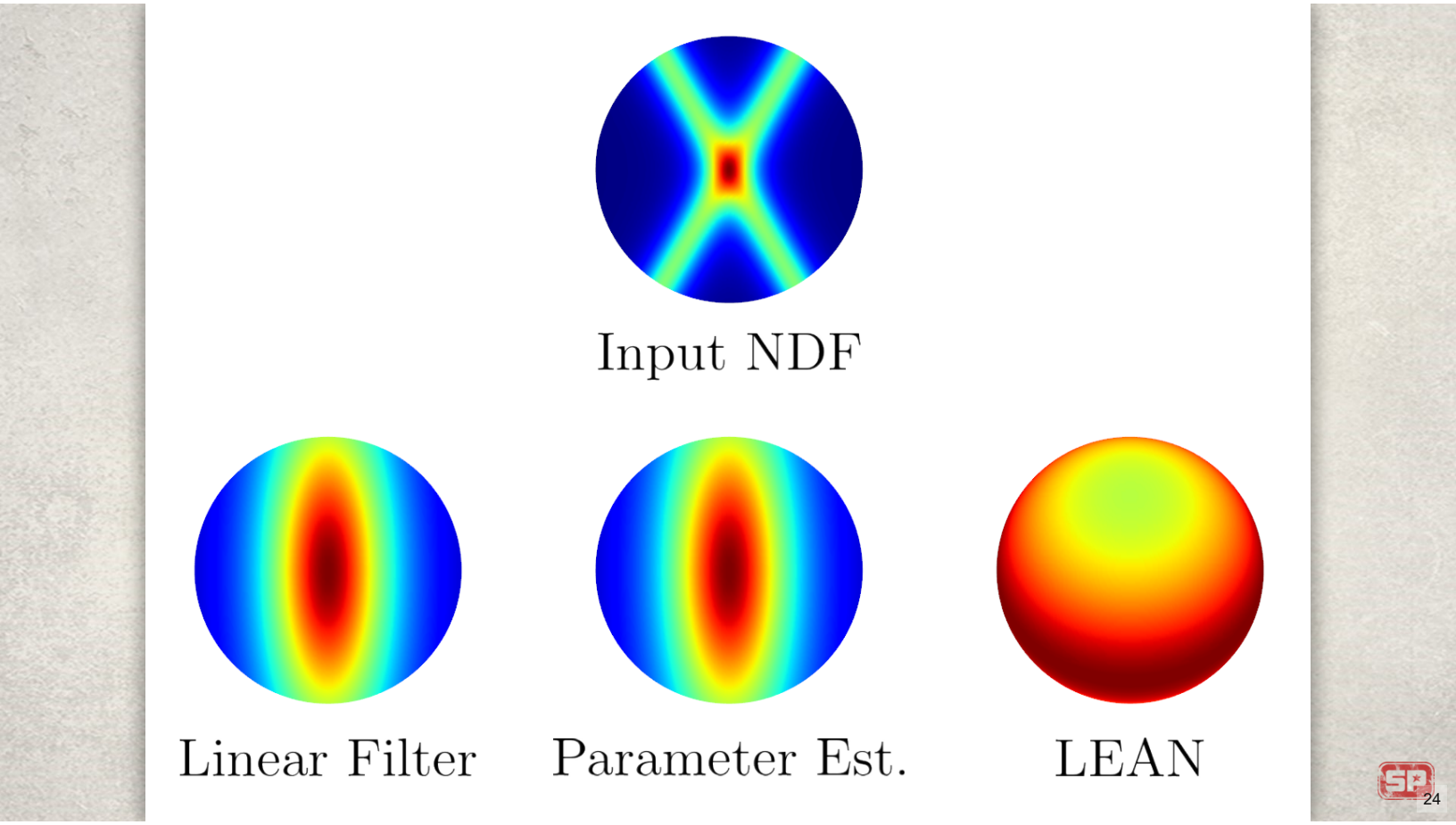
Speaker notes

This animation shows an input NDF that is a mixture of two SGGX NDFs, with the three different filtering methods on the bottom. Linear filtering and Parameter Estimation match quite closely in most cases, while LEAN filtering diverges more significantly. The first scenario that I'll point out is here.



Speaker notes

As the input NDF normals deviate from the macro-surface normal (which is the center of the sphere, since we're looking straight down the Z axis), LEAN tends to over-blur compared to the SGGX-based methods. Here the input NDF is made up of two isotropic GGX distributions (alpha_t = alpha_b = 0.2) at 45 degree angles from the Z axis.



Speaker notes

In this case, the input NDF normals are still at 45 degrees from the Z axis, but they are anisotropic "fiber-like" distributions with alpha_t = 1 (alpha_b = 0.2 as before). Note that the input NDF exhibits symmetry around the Z axis, which is captured by both linear filtering and parameter estimation, but not by LEAN.

1. Convert normal + 2x2 anisotropic GGX matrix to 3x3 SGGX matrix $\mathbf{S}$:

$$\mathbf{S} = \mathbf{M}_{\hat{\mathbf{n}}} \begin{bmatrix} S_{xx} & S_{xy} & 0 \\ S_{xy} & S_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{M}_{\hat{\mathbf{n}}}^{T}$$

where $\mathbf{M}_{\hat{\mathbf{n}}}$ rotates $\hat{\mathbf{k}} = (0, 0, 1)$ onto $\hat{\mathbf{n}} = (x, y, z)$:

$$\mathbf{M}_{\hat{\mathbf{n}}} = \begin{bmatrix} z + \frac{y^2}{1+z} & \frac{-xy}{1+z} & x \\ \frac{-xy}{1+z} & z + \frac{x^2}{1+z} & y \\ -x & -y & z \end{bmatrix}$$

Speaker notes

I'll now describe our algorithm for filtering and storage.
[next]
We first convert the normal and 2x2 anisotropic GGX matrix to a 3x3 SGGX S matrix S using this equation,
[next]
where M_n is the matrix that rotates the z axis onto n, which is given here for reference.

2. Linearly filter $\mathbf{S}$ matrices during mipmap generation
3. Extract new normal $\hat{\mathbf{n}}'$ and symmetric 2x2 submatrix from new $\mathbf{S}'$ matrix:

$$\begin{bmatrix} S'_{xx} & S'_{xy} & 0 \\ S'_{xy} & S'_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \frac{\mathbf{M}_{\hat{\mathbf{n}}'}^T \mathbf{S}' \mathbf{M}_{\hat{\mathbf{n}}'}}{\left(\mathbf{M}_{\hat{\mathbf{n}}'}^T \mathbf{S}' \mathbf{M}_{\hat{\mathbf{n}}'}\right)_{3,3}}$$

To extract $\hat{\mathbf{n}}'$ from $\mathbf{S}'$, use power iteration:

$$\hat{\mathbf{n}}'_0 = [0, 0, 1]$$

$$\hat{\mathbf{n}}'_i = \frac{\mathbf{S}' \hat{\mathbf{n}}'_{i-1}}{\left| \mathbf{S}' \hat{\mathbf{n}}'_{i-1} \right|}$$

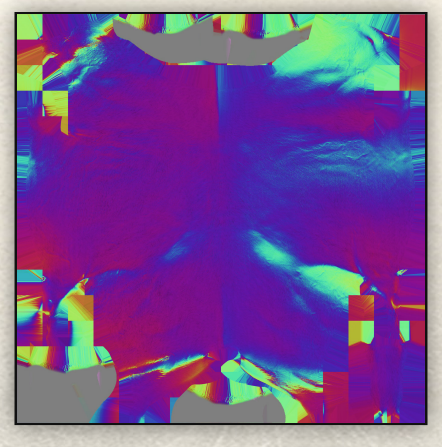
Speaker notes

We then linearly filter the S matrices during mipmap generation,
[next]
and then for each filtered matrix S' we extract a new normal n' and new 2x2 GGX matrix
[next]
We find the new normal n' using power iteration, by starting with the unit vector along the z axis and repeatedly multiplying it by S' and normalizing, which converges very quickly.

[Additional Notes]
This slide is corrected from the original version; the original slide was missing the division on the right hand side of the first equation, which ensures that the bottom right element of the matrix on the left hand side (corresponding to the projected area in the direction of the normal) is 1.

4. Store to BC5 normal map + BC7 anisotropic specular (“aniso”) map with components

$$\left[\sqrt{S'_{xx}}, \frac{1}{2}\frac{S'_{xy}}{\sqrt{S'_{xx}S'_{yy}}} + \frac{1}{2}, \sqrt{S'_{yy}}\right]$$



Speaker notes

We store the filtered normals in a BC5 texture and the 2x2 anisotropic matrix in a BC7 texture,
[next]
with this encoding, which is based on the one suggested by the SGGX paper, but for the 2x2 symmetric GGX matrix instead of the 3x3 SGGX matrix. Note that these components all have a range between 0 and 1.
[next]
This what an aniso map looks like; this is the map used for the body of the horse in the title slide.

1. Decode at run time
   - One caveat is that linear interpolation of our texture may result in a non-positive definite matrix
     - I.e., negative eigenvalues
   - Need to guard against this before using
   - We chose to extract the eigenvalues and eigenvectors
     - Need the new anisotropic tangent and bitangent for our ambient specular approximation
   - Extract α_t^2, α_b^2, $\alpha_t\alpha_b$, $\hat{\mathbf{t}}$, $\hat{\mathbf{b}}$ for use in:

$$D(\hat{\mathbf{h}}) = \frac{(\alpha_t\alpha_b)^3}{\pi\left(\left(\hat{\mathbf{t}}\cdot\hat{\mathbf{h}}\right)^2\alpha_b^2+\left(\hat{\mathbf{b}}\cdot\hat{\mathbf{h}}\right)^2\alpha_t^2+\left(\hat{\mathbf{n}}\cdot\hat{\mathbf{h}}\right)^2(\alpha_t\alpha_b)^2\right)^2}$$

Speaker notes

We then decode this texture at runtime in our shaders;
[next]
one thing we need to watch out for is that hardware filtering of our texture encoding can result in a non-positive definite matrix, so we need handle this.
[next]
We chose to extract the eigenvalues and eigenvectors, which allowed us to clamp the eigenvalues to valid ranges. We needed to do this in any case because we needed the eigenvectors for our ambient specular approximation, so this didn't incur significant overhead.
[next]
Specifically, we compute alpha_t^2, alpha_b^2, the product of alpha_t and alpha_b, as well as the tangent and bitangent, for use in the NDF shown here, as well as the shadow-masking function.

[Additional Notes]
This NDF is anisotropic GGX, rearranged slightly from other sources (e.g. Burley 2012) since the numerator and denominator have been multiplied by (alpha_t * alpha_b)^4, to eliminate some divisions.

Anisotropic Specular | Our Algorithm

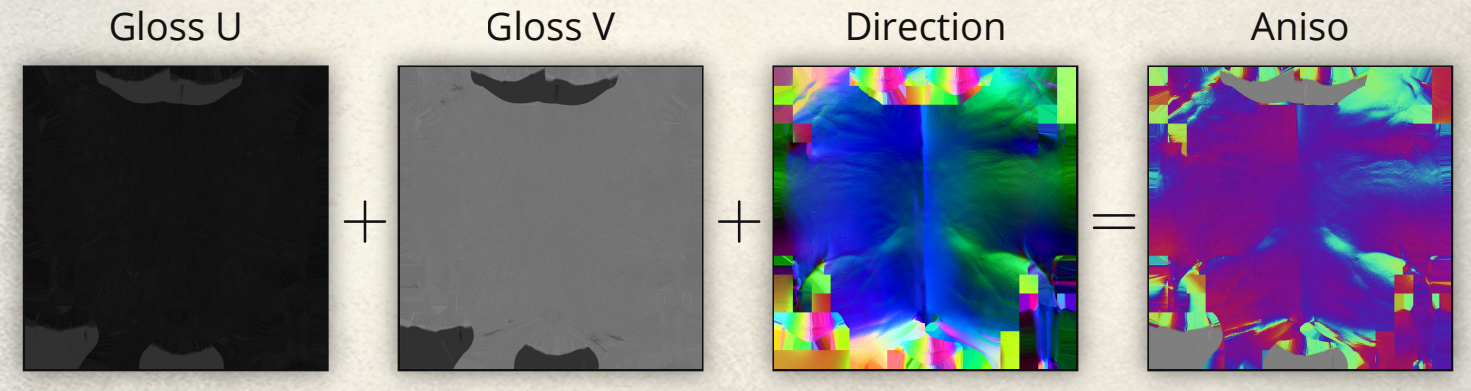
```
void DecodeSggx(float3 anisoTex, float3 tangent, float3 bitangent, out SAnisoSpecData anid)
{
    const float s_alphaMin = GgxAlphaFromGloss(1.0f);

    float sxx = Square(anisoTex.x);
    float sxy = anisoTex.x * (anisoTex.y * 2.0 - 1.0) * anisoTex.z;
    float syy = Square(anisoTex.z);

    float discr = sqrt(4.0f * Square(sxy) + Square(sxx - syy));
    float discrRcp = rcp(max(discr, 1e-6f));

    anid.m_alphaTSqr = max(saturate(0.5f * (sxx + syy + discr)), Square(s_alphaMin));
    anid.m_alphaBSqr = max(saturate(0.5f * (sxx + syy - discr)), Square(s_alphaMin));
    anid.m_alphaTB = sqrt(anid.m_alphaTSqr * anid.m_alphaBSqr);

    float cSqr = saturate(0.5f * ((sxx - syy) * discrRcp + 1.0f));
    float c = sqrt(cSqr);
    float s = Sign(sxy) * sqrt(1.0f - cSqr);

    anid.m_tangent = c * tangent + s * bitangent;
    anid.m_bitangent = -s * tangent + c * bitangent;
}
```


Speaker notes

Here's the shader code that we use to decode the texture sample
[next]
This function takes the texture sample, as well as the normal-mapped tangent and bitangent, and returns the anisotropic specular data that we need.
[next]
This value is used for clamping the alpha values to be at least the as large as alpha value corresponding to our maximum gloss value (where a gloss of 1.0 is nearly perfectly smooth, corresponding to an alpha value of ~0.001).
[next]
We then invert the texture encoding to extract the Sxx, Sxy, and Syy values;
[next]
Since we're solving for the eigenvalues of a 2x2 matrix, we're solving a quadratic equation, and this is the sqrt of the discriminant and its reciprocal.
[next]
We then solve for alpha_t squared and alpha_b square, while clamping between alpha_min squared and one, and also compute the product of alpha_t and alpha_b
[next]
Then we compute the cosine and sine of the anisotropic rotation angle
[next]
and use them to compute the anisotropic tangent and bitangent.

Anisotropic Specular | Authoring

- Aniso maps are authored by artists using a custom Substance Designer node



Speaker notes

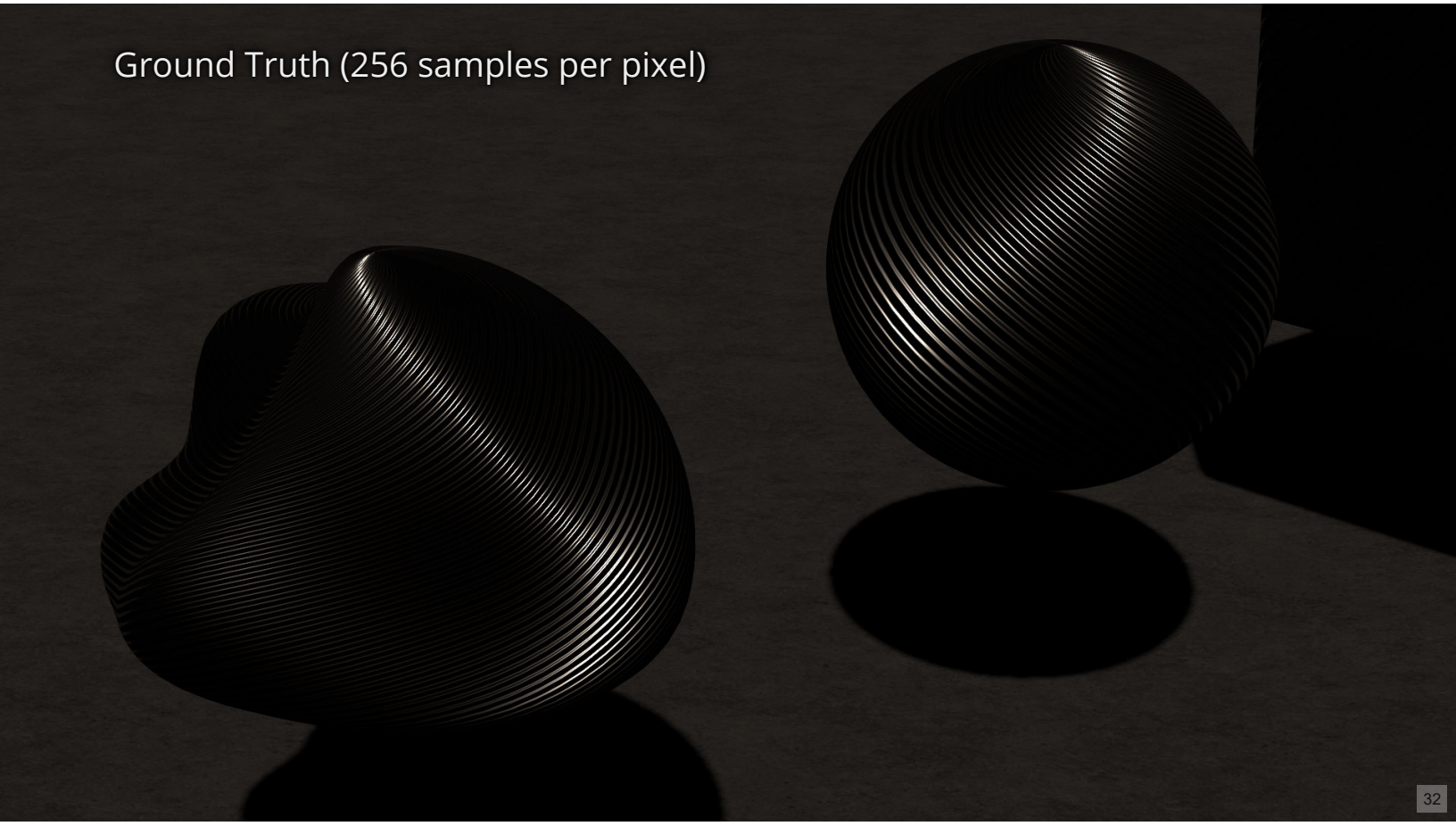
In terms of authoring our aniso maps, our artists used a custom Substance Designer node,
[next]
which takes as input two gloss maps
[next]
plus a direction map
[next]
(authored by painting a flow map in Substance Painter);
[next]
the node outputs the aniso map using the encoding that we described earlier.

Anisotropic Specular | Results

- Comparison of SGGX and LEAN filtering

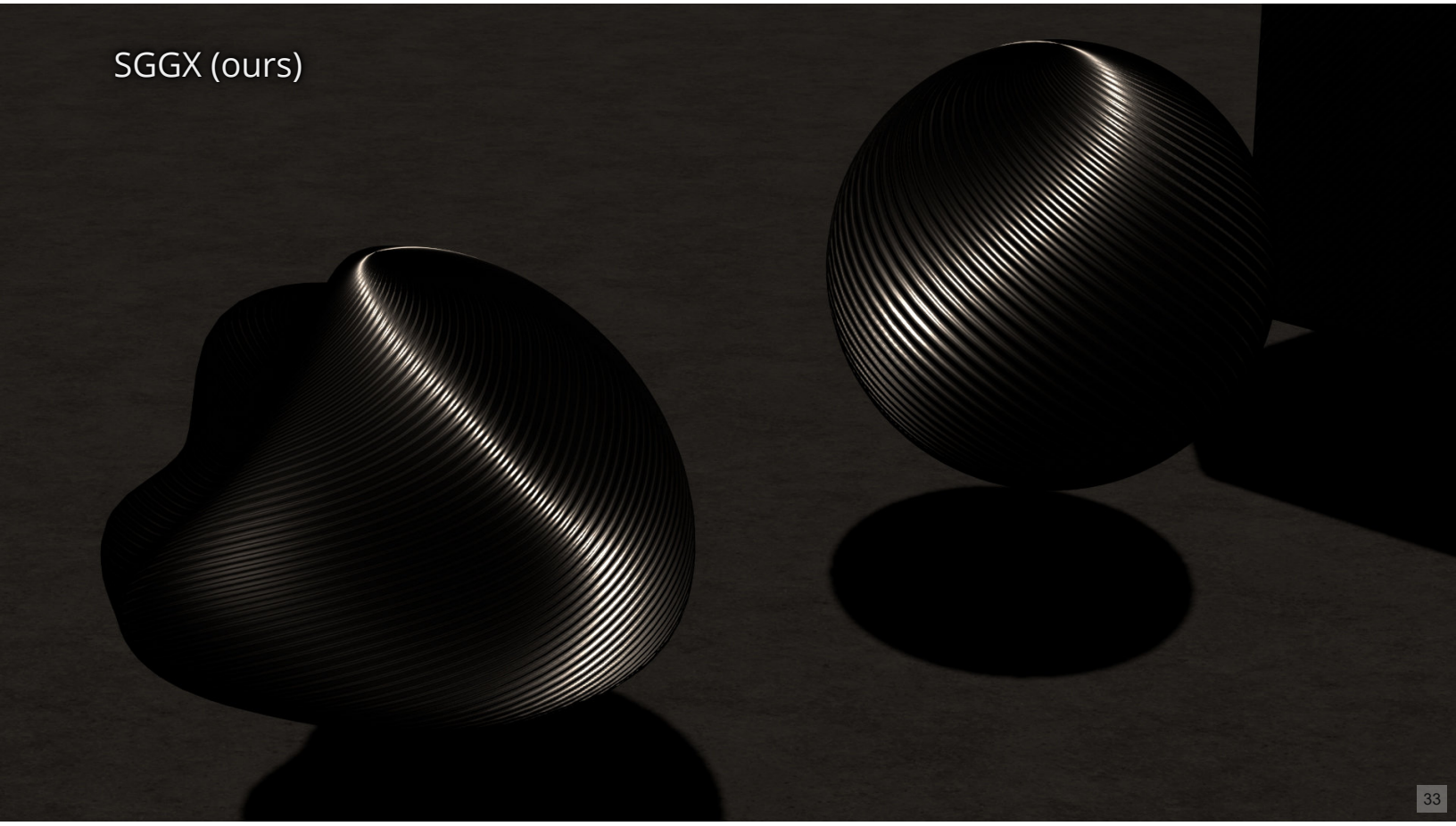
Speaker notes

I'll now show some results, which compare our SGGX-based filtering to LEAN filtering



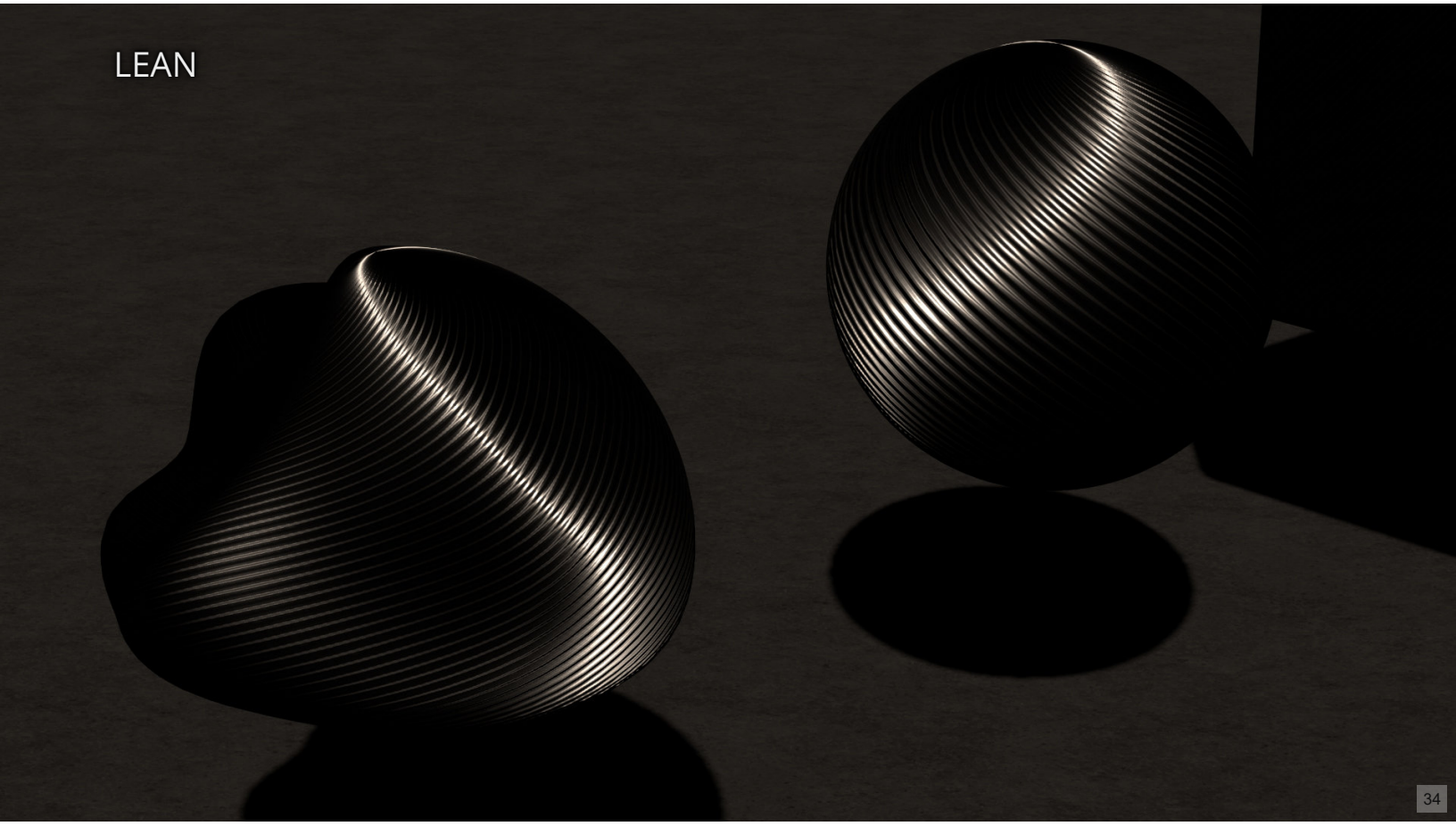
Speaker notes

This is a ground truth image, which uses only isotropic GGX specular; the anisotropy comes purely from the normal map. It was shaded using 256 samples per pixel.



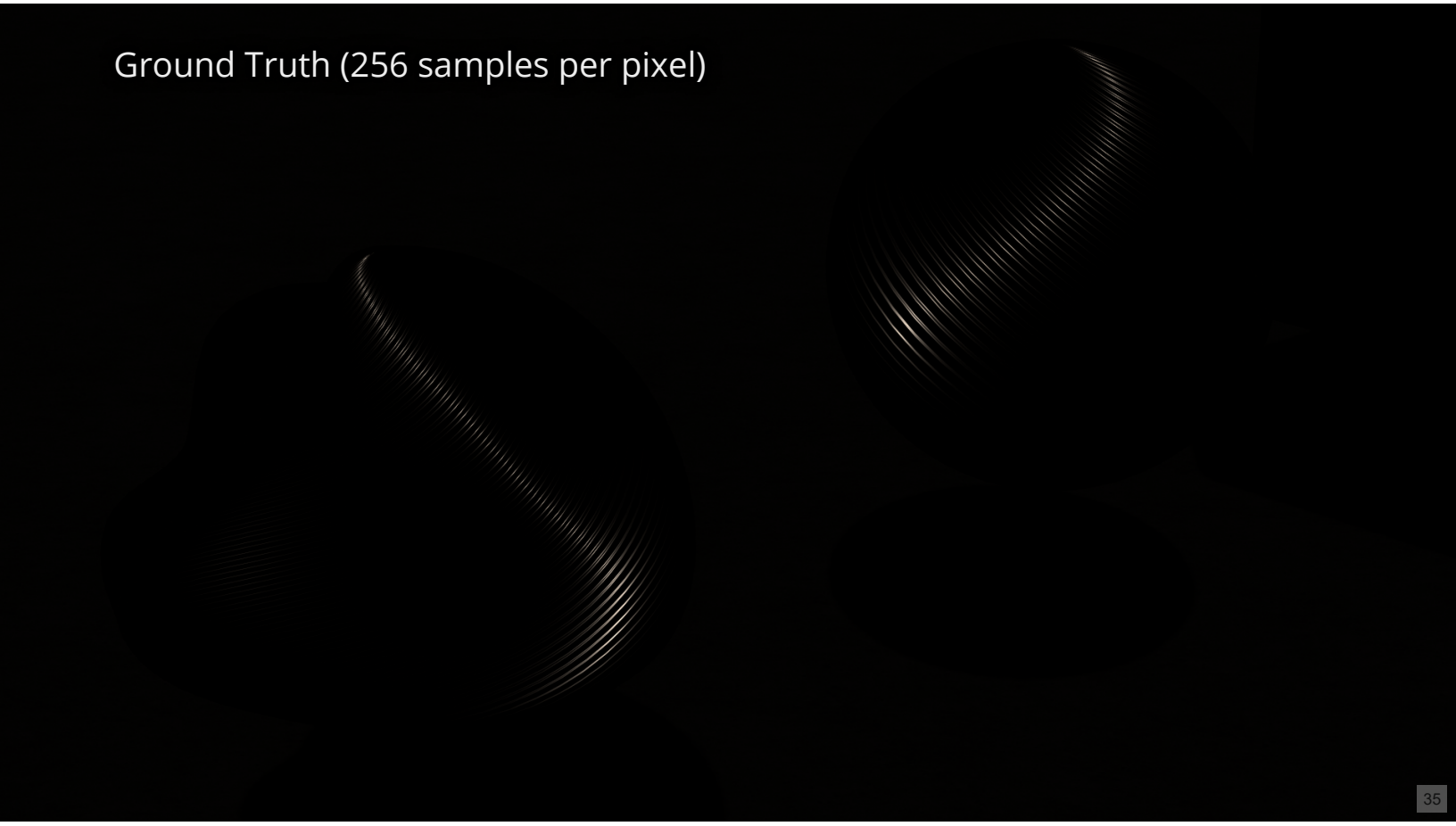
Speaker notes

Our SGGX-based filtering gives this result, which gives a highlight with approximately the same width [flip back and forth a few times]. The main differences occur at glancing angles.



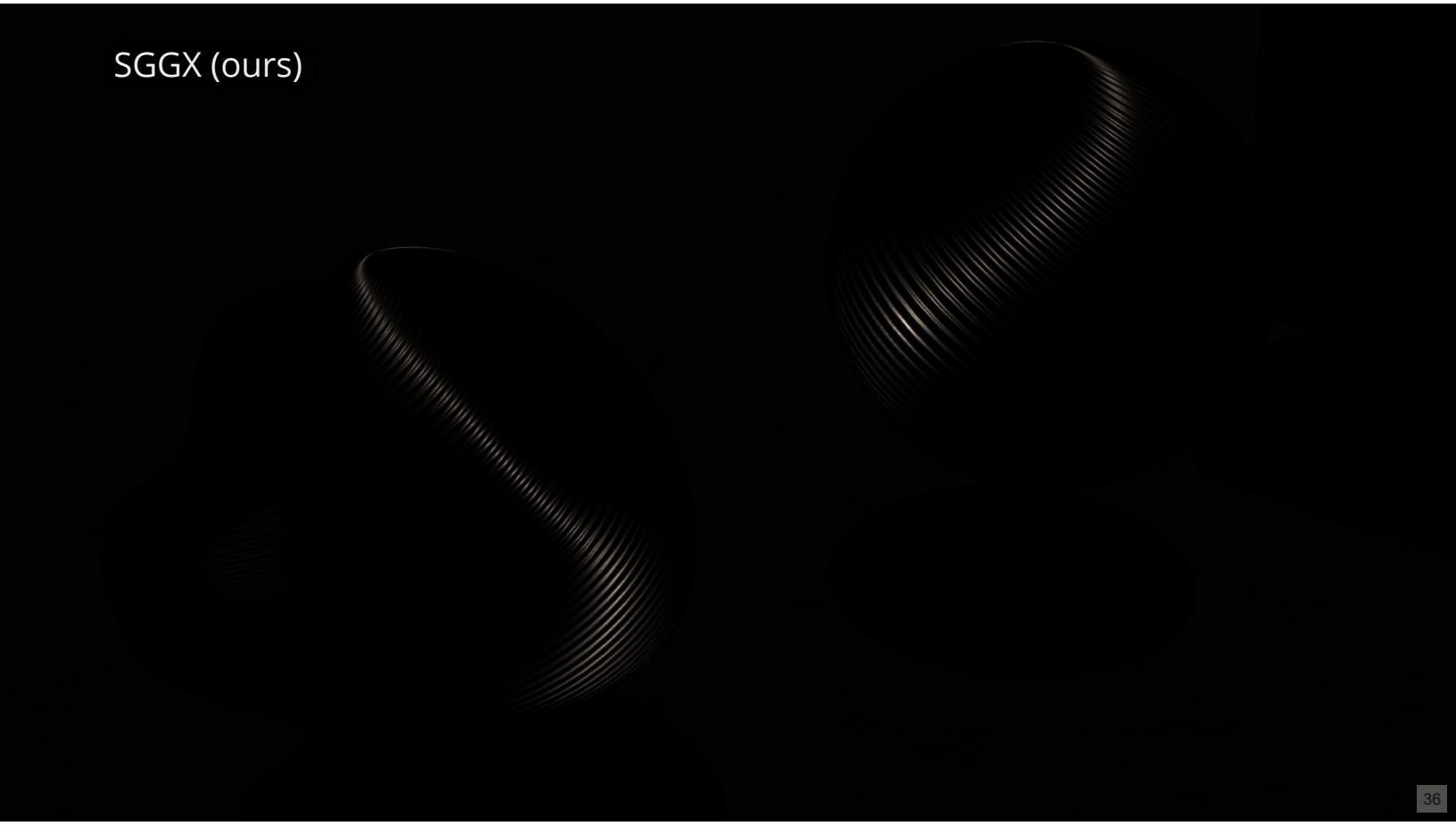
Speaker notes

LEAN filtering gives this result, which has in a wider highlight than either ground truth or SGGX.



Speaker notes

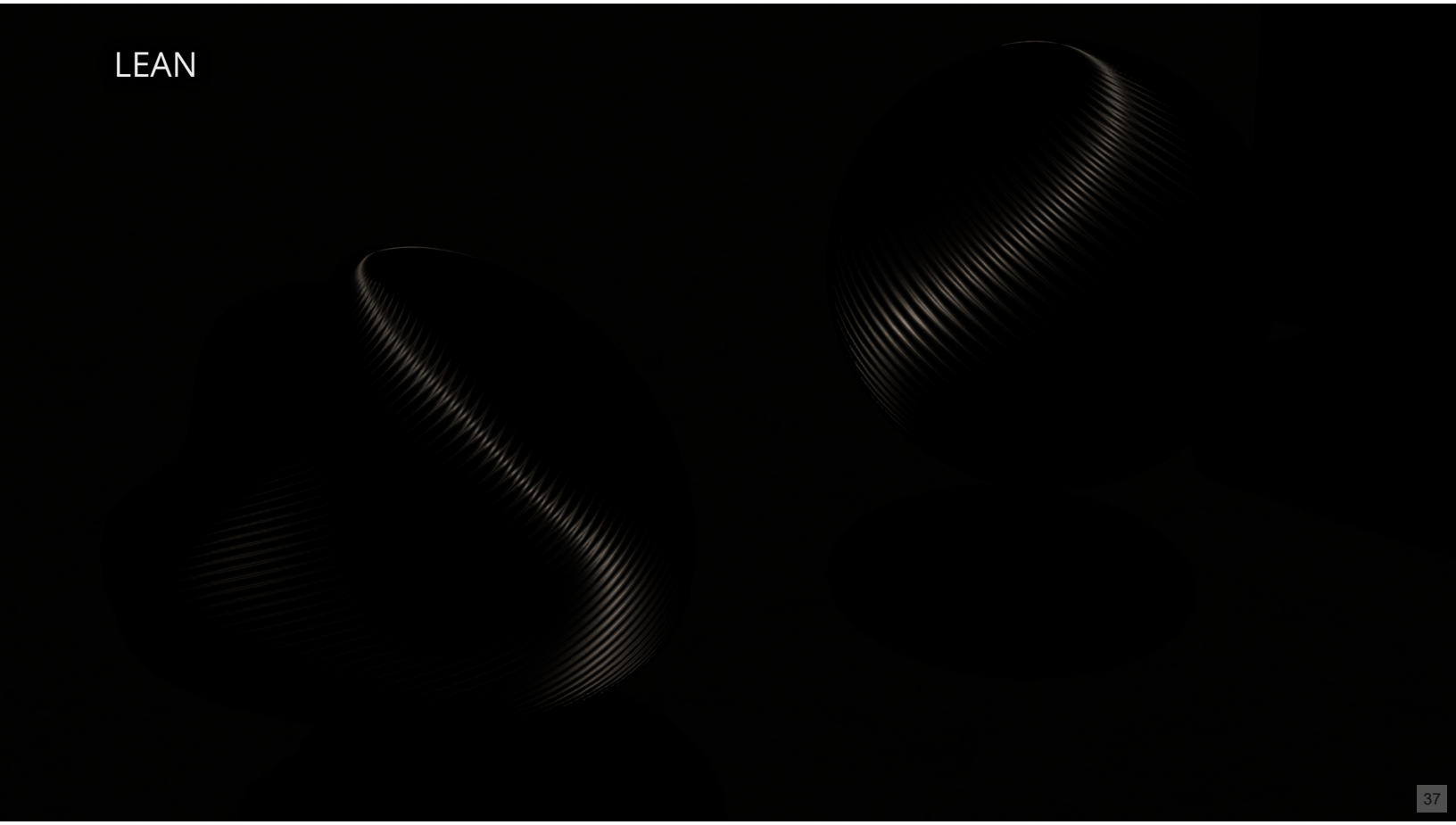
This is the same scene as before, but with the exposure reduced to avoid clipping the highlight.



Speaker notes

The brightness of the SGGX highlight is only slightly reduced compared to the ground truth...

LEAN



Speaker notes

...while the LEAN highlight is noticeably dimmer, especially at the brightest parts of the highlight.

Speaker notes

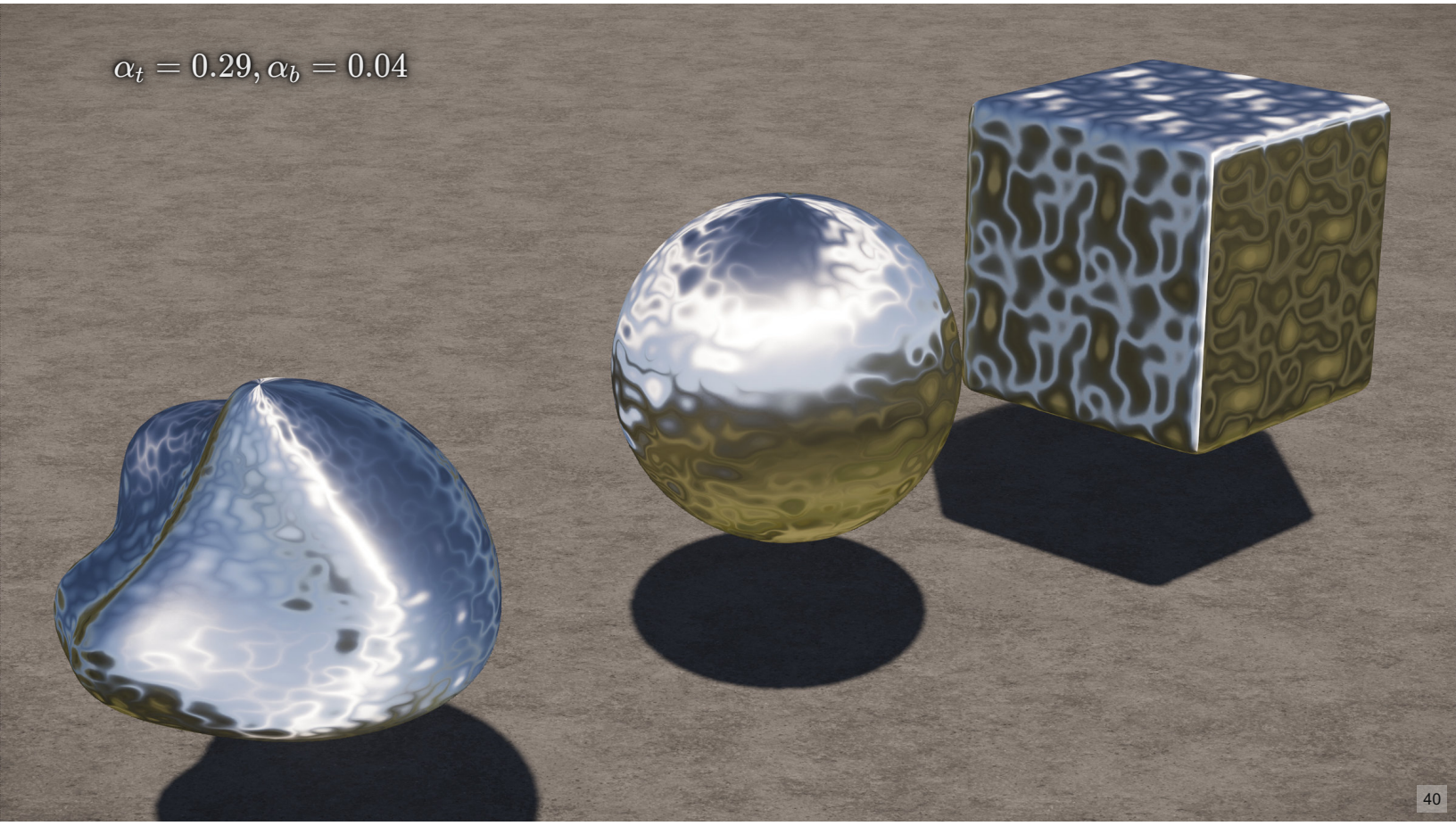
This video shows how our anisotropic maps behave as the mip level increases. Shading is quite stable, and in general the appearance remains consistent at multiple scales.

Anisotropic Specular | Limitations

- Due to encoding, texture hardware does not interpolate $\mathbf{S}$ matrix directly, which may result in artifacts
  - In practice, for real-world materials, this did not result in problems
- Combinations of large and small α_t and α_b values do not work well

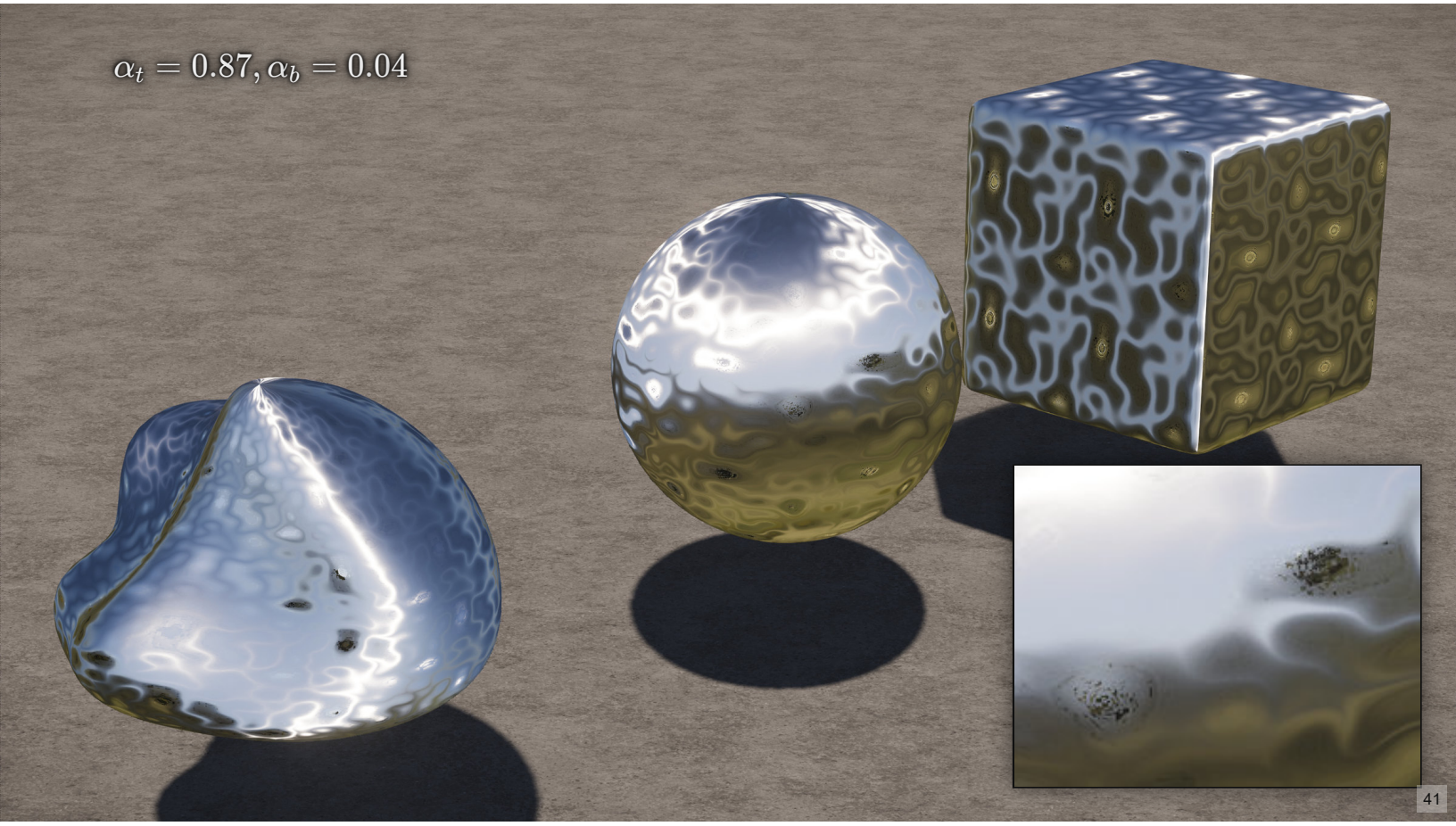
Speaker notes

There are some limitations to this approach, primarily related to the texture encoding. Since we're not directly interpolating the S matrix values which can result in artifacts. In practice, for real-world materials, we didn't encounter problems with this -- clamping the alpha values appears to work well here.
[next]
Precision issues can arise when using combinations of large and small alpha_t and alpha_b values, resulting in artifacts; I'll demonstrate with an example.



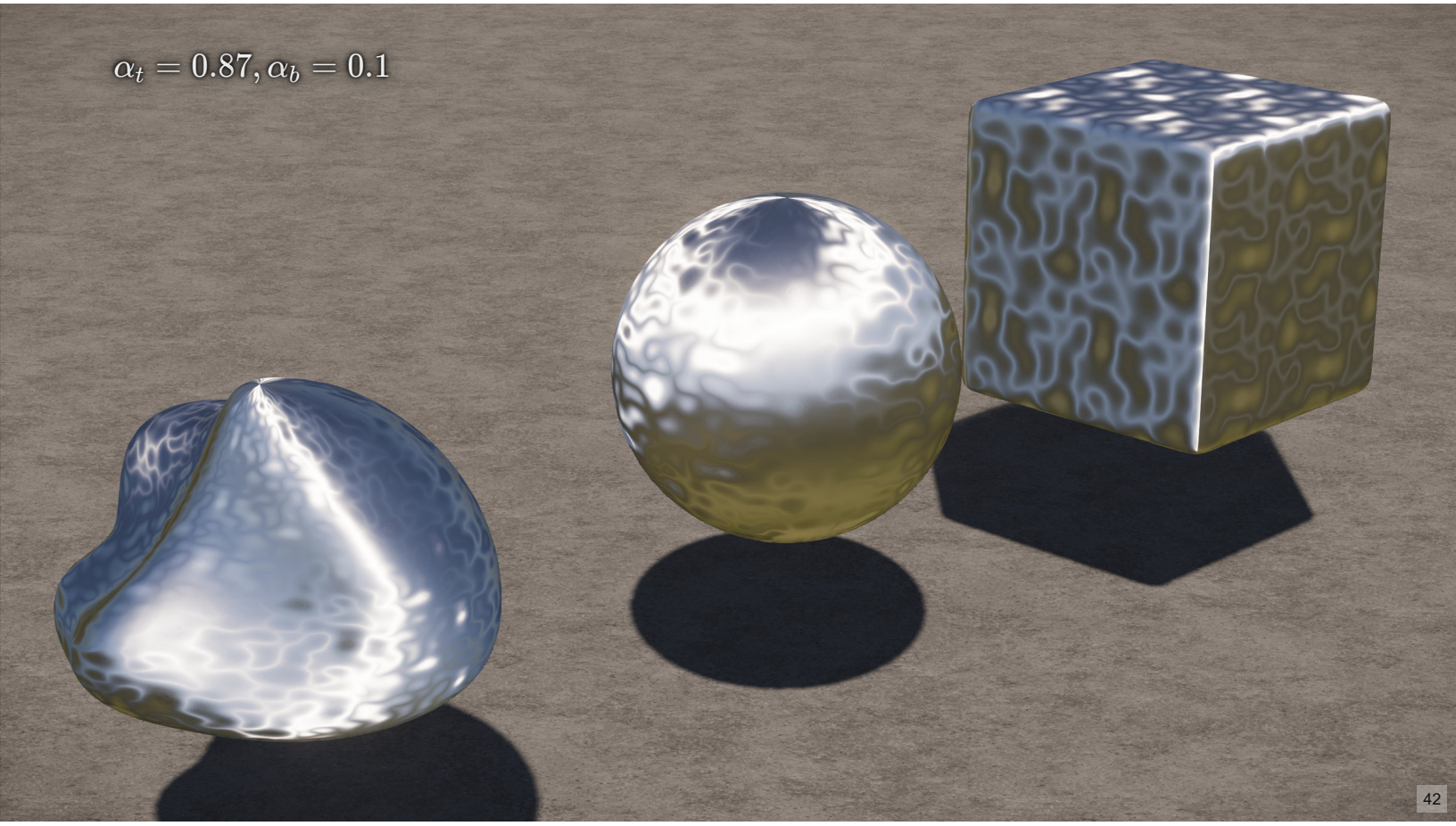
Speaker notes

Here, there are no artifacts, using alpha_t of 0.29 and alpha_b of 0.04.



Speaker notes

If we increase alpha_t to 0.87, keeping alpha_b constant, we see some artifacts,
[next]
show magnified here.



Speaker notes

These artifacts do go away if alpha_b increases from 0.04 to 0.1. In practice we didn't encounter these artifacts for real-world materials, so we haven't spent any time trying to address this, but it's something to be aware of.

Anisotropic Specular | Summary

- New SGGX-based filtering approach
  - Simple to implement
  - Same number of parameters (5) as LEAN
  - Same/similar shading cost
  - Good results for strong normals and highly anisotropic NDFs
- New anisotropic roughness texture encoding
  - Based on SGGX encoding
  - Works well with BC7 compression
    - 1/4 to 1/8th the size of LEAN encoding

Speaker notes

In summary, I've presented a new SGGX-based filtering approach
[next]
that is simple to implement;
[next]
it has the same number of parameters as LEAN;
[next]
and has roughly the same shading cost;
[next]
and has produced good results for cases with strong normals and highly anisotropic NDFs.
[next]
I've also shown a new texture encoding for anisotropic roughness, based on the SGGX encoding
[next]
which works well with BC7 compression and results in textures that are 1/4 to 1/8th the size of the LEAN encoding, depending on the bit depth used.

[Additional Notes]
These developments are more or less orthogonal to each other; i.e., it would be possible to use the LEAN filtering approach with our encoding, or vice versa.

Speaker notes

Next, I'll talk about our new anisotropic asperity scattering BRDF.

Anisotropic Asperity Scattering BRDF

- Asperity scattering
  - ... AKA sheen
  - ... AKA fuzziness
  - ... AKA fuzz

Speaker notes

Because that's a mouthful
[next]x5

Anisotropic

- Asperity scattering
  - … AKA sheen
  - … AKA fuzziness
  - … AKA fuzz

Speaker notes

Anisotropic Fuzz

- Asperity scattering
  - … AKA sheen
  - … AKA fuzziness
  - … AKA fuzz

Speaker notes

I'm going to call it anisotropic fuzziness or fuzz, which is what we call it at Sucker Punch.
[next]
Here we have a canonical example of the effect of fuzziness, a peach vs a nectarine,
[next]
and it's clear that the luminance of the peach is much more uniform when front-lit, compared to the nectarine which has a strong Lambertian falloff.

Anisotropic Fuzz

- Asperity scattering
  - … AKA sheen
  - … AKA fuzziness
  - … AKA fuzz
- New BRDF was researched during look development

Speaker notes

This new BRDF was researched during look development,

Anisotropic Fuzz

- Asperity scattering
  - … AKA sheen
  - … AKA fuzziness
  - … AKA fuzz
- New BRDF was researched during look development
- Realistic velvet, including crushed velvet, was important at the time
- In Ghost, used for moss, felt, horse coats, and some cloth.

Speaker notes

and at that stage of the project, back in early 2015,
[next]
realistic velvet, including the anisotropic behavior exhibited by crushed velvet, were important, although changes in the project's setting meant that velvet like materials didn't feature in the shipping game.
[next]
This BRDF *was* used in Ghost for other materials such as moss, felt, horse coats, and some cloth.

Anisotropic Fuzz | Previous Work

GHOST

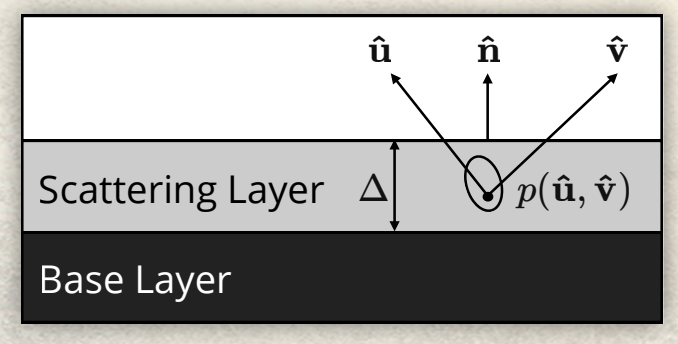
- Our old model was ad hoc: Fresnel-like term + wrap lighting
- Neubelt & Pettineo (Ready at Dawn) presented their model in this course in 2013
  - Uses microfacet framework
  - Energy conserving
- *The Secret of Velvety Skin* (Koenderink and Pont, 2002)
  - Models fuzziness as a thin single-scattering layer
- No existing models supported anisotropy, to our knowledge

Speaker notes

In terms of previous work, we had an old model which was ad-hoc, and didn't meet the quality bar that we wanted to achieve with Ghost.
[next]
In this course in 2013, Neubelt and Pettineo presented their model, which is a kind of inverted Gaussian, and is derived using the microfacet framework.
[next]
One paper that got our attention was "The Secret of Velvety Skin" by Koenderink and Pont from 2002; it models fuzziness as a thin single-scattering layer on top of a conventional base layer.
[next]
But none of the existing models supported anisotropy, which is necessary to accurately reproduce materials like crushed velvet, so we decided to develop our own BRDF.

Anisotropic Fuzz | New Model

- Our new model was inspired by Koenderink and Pont's model
- Extended to support:
  - Anisotropy
  - Shadowing of base layer
  - Spherical harmonic lighting



- $\hat{\mathbf{u}}$: Light direction
- $\hat{\mathbf{v}}$: View direction
- $p(\hat{\mathbf{u}}, \hat{\mathbf{v}})$: Phase function
- Δ: Thickness of scattering layer
- λ: Mean free path
- $d = \frac{\Delta}{\lambda} \ll 1$: Density

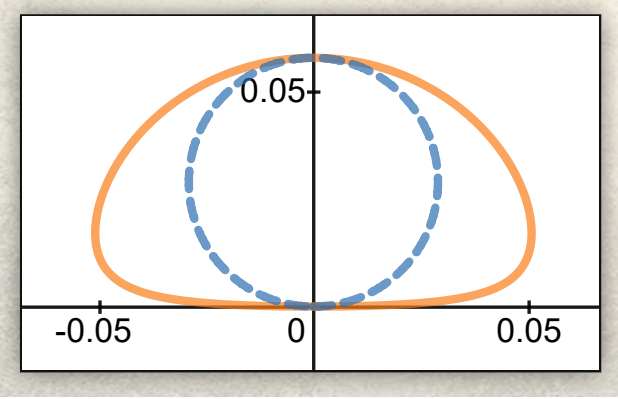
Speaker notes

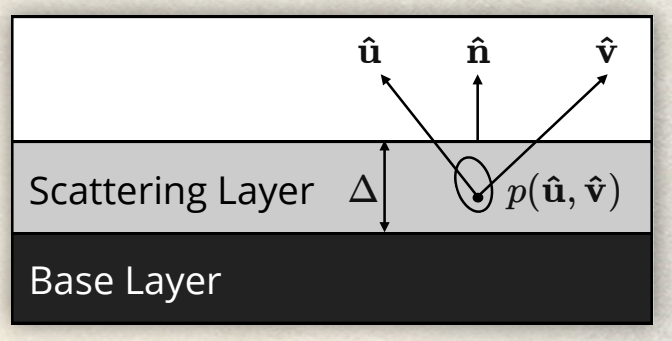
Our model was based on Koenderink and Pont's model,
[next]
and extended to support anisotropy, with shadowing of the base layer in order to conserve energy. We also derived a form suitable for use with spherical harmonic lighting, which is what we use for diffuse indirect lighting. This ensures that the appearance remains consistent in all lighting scenarios.
[next]
Here's a schematic of the model; we have a conventional base layer with a scattering layer of thickness delta on top, with surface normal n;
[next]
u is the light direction;
[next]
p is the phase function;
[next]
v is the view direction;
[next]
and if lambda is the mean free path,
[next]
then we define d to be delta over lambda, which we call density, and which is assumed to be much less than 1.

- Koenderink and Pont show that the BRDF of the scattering layer is:

$$f(\hat{\mathbf{u}}, \hat{\mathbf{v}}) = \mathbf{c}_f p(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \frac{\left(1 - e^{-d\frac{(\hat{\mathbf{u}}+\hat{\mathbf{v}})\cdot\hat{\mathbf{n}}}{(\hat{\mathbf{u}}\cdot\hat{\mathbf{v}})(\hat{\mathbf{v}}\cdot\hat{\mathbf{n}})}}\right)}{(\hat{\mathbf{u}}+\hat{\mathbf{v}})\cdot\hat{\mathbf{n}}}$$

- Isotropic case:





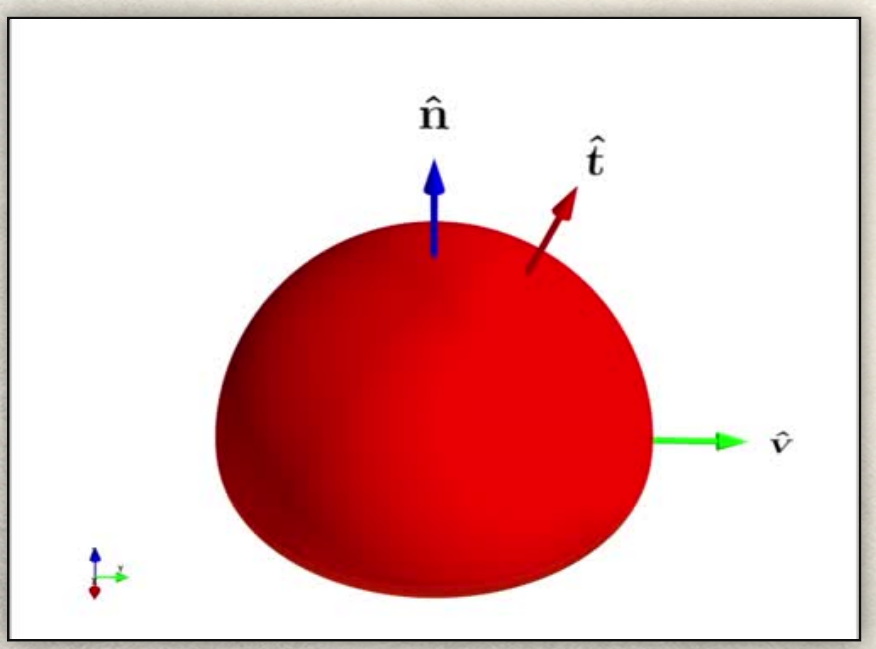
- $\hat{\mathbf{u}}$: Light direction
- $\hat{\mathbf{v}}$: View direction
- $p(\hat{\mathbf{u}}, \hat{\mathbf{v}})$: Phase function
- Δ: Thickness of scattering layer
- λ: Mean free path
- $d = \frac{\Delta}{\lambda} \ll 1$: Density
- $\mathbf{c}_f$: Albedo

Speaker notes

Under these assumptions,
[next]
the BRDF of the scattering layer is given by this,
[next]
where c_f is the albedo of the scattering layer.
[next]
In the isotropic scattering case, where the phase function is constant, the plot of the BRDF times the cosine term looks like the orange curve here; the dashed blue curve is the is cosine lobe. One thing to note is that this curve is nearly constant over the hemisphere; we'll make use of this later.

[Additional Notes]
Click on the graph in the lower left to open an interactive version, or use this URL: https://www.desmos.com/calculator/icwwbkbtkw

- Strongly dependent on view direction

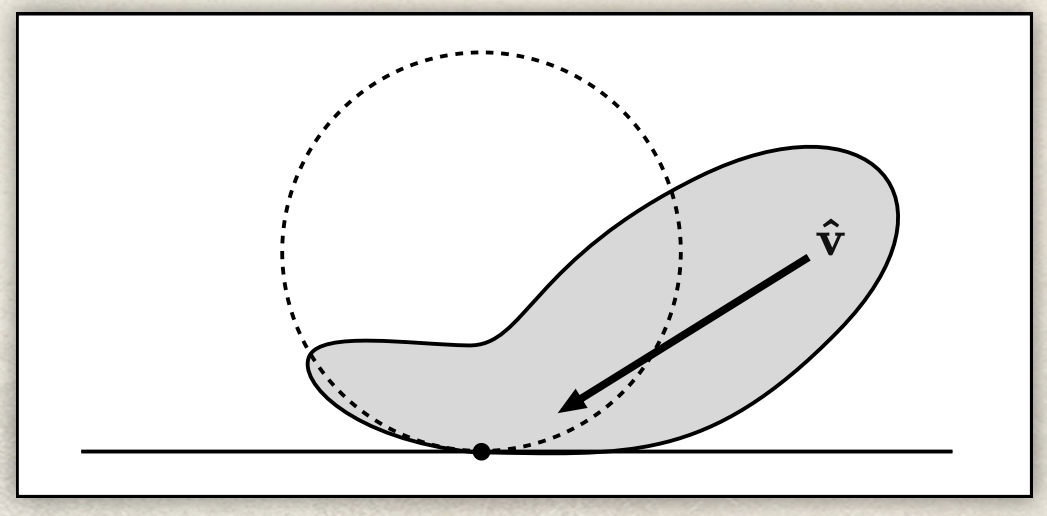


Speaker notes

The amplitude of the scattering is also very strongly dependent on the view direction, which can be seen here, and matches what we observe for a material like velvet.

Anisotropic Fuzz | Velvet Scattering

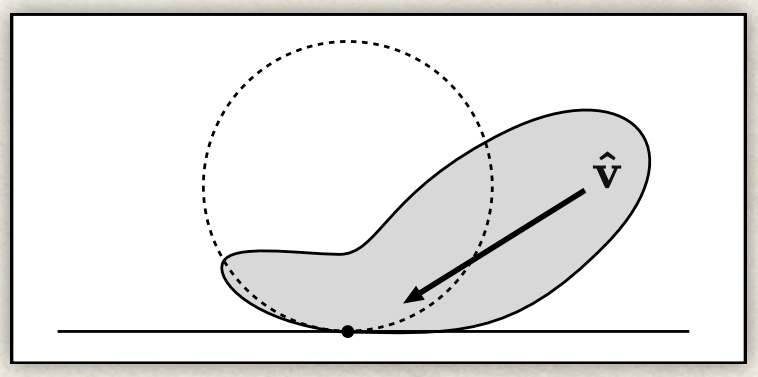
- According to Koenderink and Pont, the scattering diagram for black velvet resembles:



Speaker notes

In their paper, Koenderink and Pont present a scattering diagram for black velvet which looks like this;

- According to Koenderink and Pont, the scattering diagram for black velvet resembles:



- Strong back scattering lobe
- Smaller forward scattering lobe
- Can we find a phase function that reproduces this?

Speaker notes

it has a strong back-scattering lobe
[next]
and a smaller forward-scattering lobe;
[next]
we wanted to try to find a phase function that could reproduce this.

- Starting with the Schlick approximation to the Henyey-Greenstein phase function:

$$p_{\text{Schlick}}(k, \cos\theta) = \frac{1}{4\pi} \frac{1-k^2}{(1-k\cos\theta)^2}$$

  - k: asymmetry parameter
- Make it dependent on the angle between the half vector $\hat{\mathbf{h}}$ and the fiber direction $\hat{\mathbf{t}}$:

$$p_V(k, \hat{\mathbf{h}}, \hat{\mathbf{t}}) = r(k) p_{\text{Schlick}}(k, \sin(\hat{\mathbf{t}}, \hat{\mathbf{h}}))$$

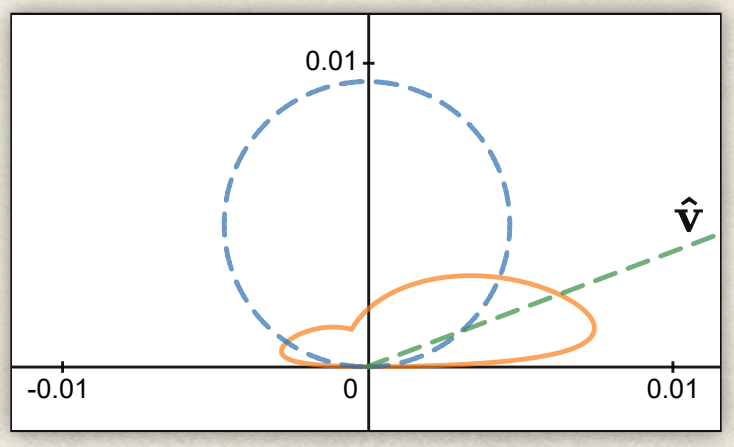
  - $r(k)$: normalization factor (approximate and conservative)

Speaker notes

We started with the Schlick approximation to the Henyey-Greenstein phase function,
[next]
and, inspired by the Kajiya-Kay hair rendering model, we modified it to make it dependent on the sine of the angle between the half-vector and the fiber direction t. This breaks the normalization of the phase function, so we added an extra term, r, to correct for this. Since we didn't find an analytical form for this, we used an approximate and conservative form, which resulted in some energy loss.

[Additional Notes]
We exposed Schlick k as a "Spread" parameter, with k = -0.99 * (1.0 - spread).

- Using this function in the BRDF, we get:



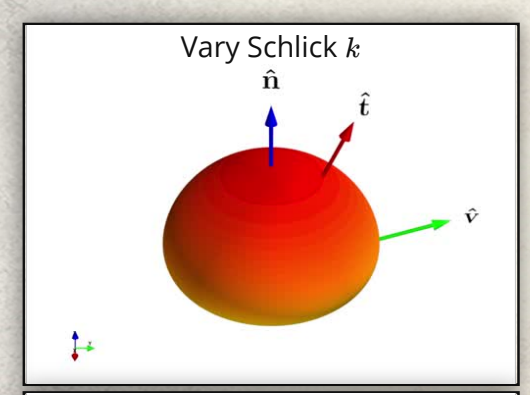
- ... which is qualitatively similar, at least.

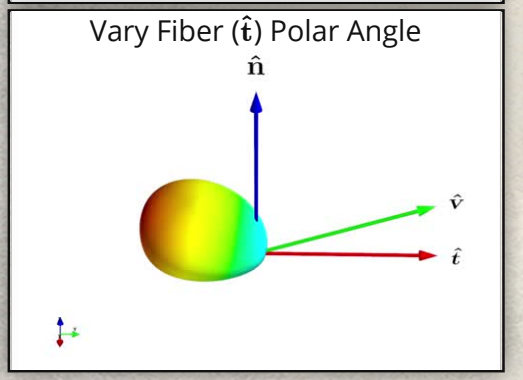
Speaker notes

Using this new phase function in the BRDF,
[next]
we get a scattering diagram that looks like this,
[next]
which has the same basic shape as the velvet scattering diagram.

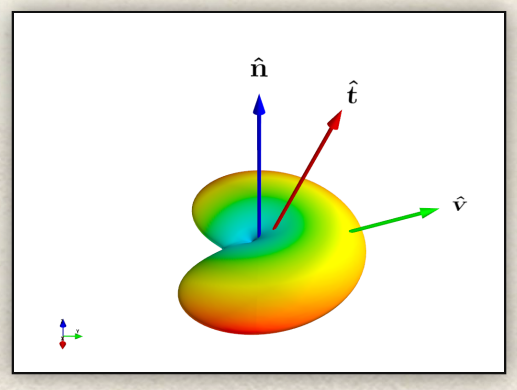
[Additional Notes]
Click on the graph to open an interactive version, or use this URL: https://www.desmos.com/calculator/hwp7tga0vk

Anisotropic Fuzz | Phase Function

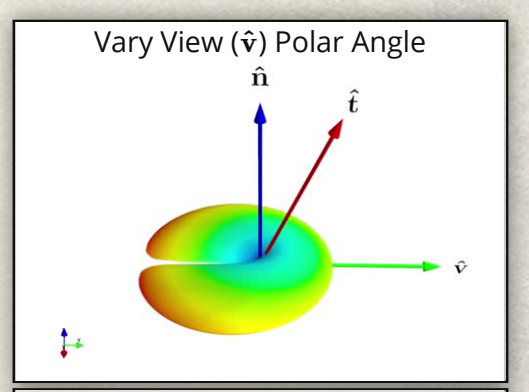


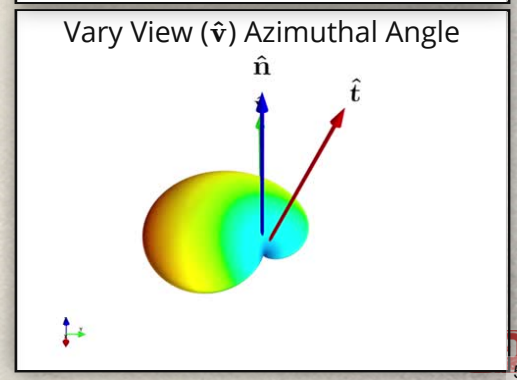


3D Plot:



(Note: Normalized Amplitude)





Speaker notes

In 3D it looks like this,
[next]
and here's what it looks like if we animate the Schlick k parameter,
[next]
the fiber polar angle,
[next]
the view polar angle,
[next]
and the view azimuthal angle.
Note the strong tangential scattering in many configurations. Also note that these plots have normalized amplitude; in reality their scale changes considerably.

- More recent work: use a “fiber-like” specular SGGX phase function
  - Analytical normalization

$$D_F\left(\hat{\mathbf{h}}\right) = \frac{\alpha^3}{\pi\left(\left(\hat{\mathbf{h}}\cdot\hat{\mathbf{t}}\right)^2(1-\alpha^2)+\alpha^2\right)^2}$$

$$\sigma_F\left(\hat{\mathbf{u}}\right) = \sqrt{1-\left(\hat{\mathbf{u}}\cdot\hat{\mathbf{t}}\right)^2\left(1-\alpha^2\right)}$$

$$p_F\left(\hat{\mathbf{u}},\hat{\mathbf{v}}\right) = \frac{D\left(\hat{\mathbf{h}}\right)}{4\sigma(\hat{\mathbf{u}})}$$

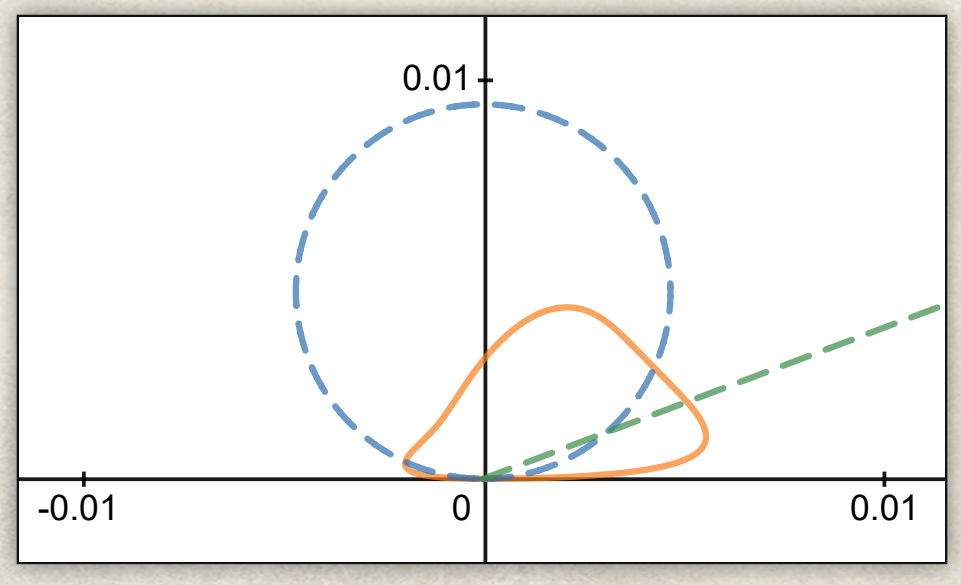
- $\hat{\mathbf{u}}$: Light direction
- $\hat{\mathbf{v}}$: View direction
- $\hat{\mathbf{h}} = \frac{\hat{\mathbf{u}}+\hat{\mathbf{v}}}{|\hat{\mathbf{u}}+\hat{\mathbf{v}}|}$: Half vector
- $\hat{\mathbf{t}}$: Fiber direction
- α: GGX roughness
- D_F: (S)GGX “fiber” NDF
- σ_F: Microflake projected area
- $p_F(\hat{\mathbf{u}},\hat{\mathbf{v}})$: Phase function

Speaker notes

In more recent work, after shipping Ghost, we wanted to try to solve the energy loss problem, so instead of the Schlick-based phase function, we decided to try a fiber-like SGGX phase function,
[next]
which has analytic normalization.
[next]
The phase function is composed of two parts; the first is the NDF shown here. Alpha is the GGX roughness, or the projected area of the microflakes in the fiber direction t; the projected area perpendicular to t is 1, which is what makes it a "fiber-like" distribution.
[next]
The second term in the phase function is the microflake projected area in the light direction u, given by this.
[next]
The phase function is then the NDF divided by four times the projected area.

[Additional Notes]
In terms of shader code, this is about the same cost as the previous phase function. We compute alpha from our shader "spread" parameter using:
alpha = sqrt(spread) * 0.85 + 0.15,
which gives a similar response compared to the Schlick-based phase function.

- Using the SGGX phase function we get:

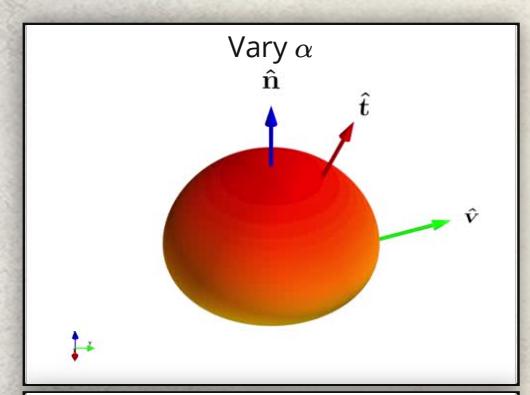


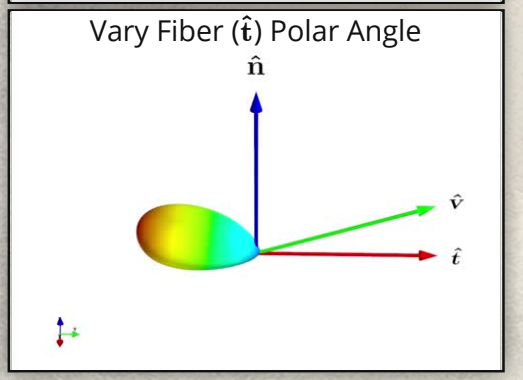
Speaker notes

The scattering diagram using this phase function looks like this, which is slightly blobbier than the Schlick version, but still has a strong back-scattering component with a smaller forward scattering component. In practice they are very similar in appearance, with SGGX being slightly brighter due to exact energy conservation.

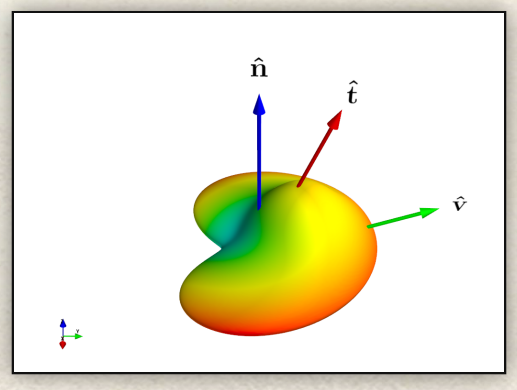
[Additional Notes]
Click on the graph to open an interactive version, or use this URL: https://www.desmos.com/calculator/botqtjpnus

Anisotropic Fuzz | Phase Function

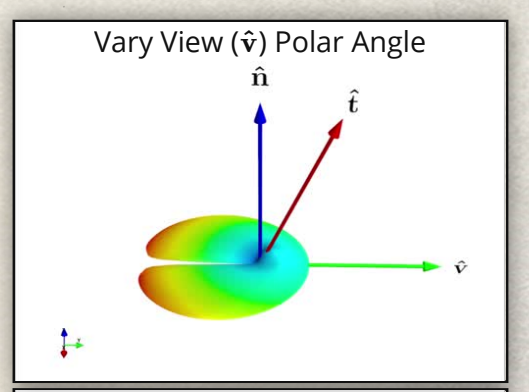


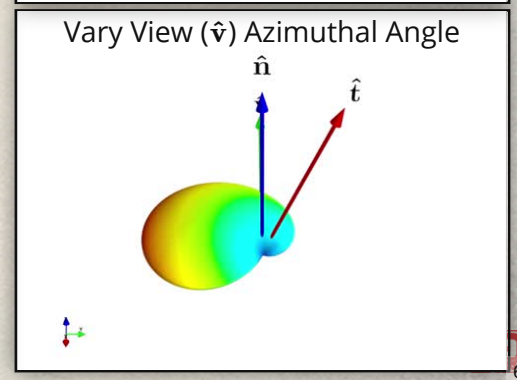


3D Plot:



(Note: Normalized Amplitude)





Speaker notes

In 3D it looks like this,
[next]
and again animating parameters starting with the fiber projected area,
[next]
the fiber polar angle,
[next]
the view polar angle,
[next]
and the view azimuth angle. Note that there is a stronger reflection when the light is aligned with the fiber direction compared to our Schlick-based model.

- Need to account for shadowing of base layer by scattering layer
- Don't account for all interactions to keep things simple(r)
- Probability of a ray penetrating to the base layer without scattering is:

$$P_p(\hat{\mathbf{u}}) = e^{-\frac{d}{\hat{\mathbf{u}}\cdot\hat{\mathbf{n}}}}$$

- Probability of a ray reflected by the base layer escaping is:

$$P_e(\hat{\mathbf{v}}) = e^{-\frac{d}{\hat{\mathbf{v}}\cdot\hat{\mathbf{n}}}}$$

- Probability that an incident ray is scattered towards the base layer is:

$$P_s(\hat{\mathbf{u}}) = 1 - \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}})\, \mathrm{d}\omega_{\hat{\mathbf{v}}}$$

Speaker notes

In order to conserve energy we need to account for how much incident light is transmitted to the base layer, and also how much light is scattered on the way out towards the eye.
[next]
We're only going to account for the dominant interactions here;
[next]
The first term to consider, is the probability that a ray penetrates through the scattering layer, to the base layer; we call that P_p, given by this;
[next]
the probability that a ray, reflected by the base layer, escapes through the scattering layer in direction v is given by this; we call this P_e;
[next]
finally, we call P_s the probability that a ray, scattered by the scattering layer, is scattered towards the base layer, and not away from the material. It's equal to one minus the integral of the phase function over the hemisphere. Rays scattered away from the material are accounted for by the scattering layer BRDF; this accounts for the rest.

$$P_p(\hat{\mathbf{u}}) = e^{-\frac{d}{\hat{\mathbf{u}}\cdot\hat{\mathbf{n}}}} \qquad P_e(\hat{\mathbf{v}}) = e^{-\frac{d}{\hat{\mathbf{v}}\cdot\hat{\mathbf{n}}}} \qquad P_s(\hat{\mathbf{u}}) = 1 - \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}})\,\mathrm{d}\omega_{\hat{\mathbf{v}}}$$

- Combining these, we get the following for the base layer attenuation:

$$\mathbf{c}_{\text{atten}}(\hat{\mathbf{u}}, \hat{\mathbf{v}}) = \left[P_p(\hat{\mathbf{u}}) + \mathbf{c}_f(1 - P_p(\hat{\mathbf{u}}))P_s(\hat{\mathbf{u}})\right] P_e(\hat{\mathbf{v}})$$

- No analytical expression for P_s (even for SGGX 😔)
- Use the following approximation:

$$P_s(\hat{\mathbf{u}}) \approx c_0 + c_1\hat{\mathbf{u}}\cdot\hat{\mathbf{n}} + c_2\hat{\mathbf{u}}\cdot\hat{\mathbf{t}} + c_3(\hat{\mathbf{u}}\cdot\hat{\mathbf{n}})^2 + c_4(\hat{\mathbf{u}}\cdot\hat{\mathbf{n}})(\hat{\mathbf{u}}\cdot\hat{\mathbf{t}}) + c_5(\hat{\mathbf{u}}\cdot\hat{\mathbf{t}})^2$$

Speaker notes

[next]
We can combine these three terms into a base layer attenuation factor, which is equal to
[next]
P_p, the probability of penetrating to the base layer, plus the fiber color times one minus P_p,
[next]
times P_s, the probability of being scattered down towards the base layer,
[next]
all multiplied by P_e, the probability of escaping through the scattering layer towards the eye in direction v.
[next]
Unfortunately, there's no analytical expression for P_s,
[next]
so we use the following approximation, with the constants c_i's determined using curve fitting.

[Additional Notes]
This approximation for P_s(u) must be evaluated at runtime (in the shader), so it needs to be relatively inexpensive. These c_i's also depend on Schlick k (or GGX alpha), as well as the fiber tilt angle. We further fit curves to the curve constants to make this less expensive to compute at shader compile time); a LUT could also be used. Please see the Supplemental Material for details.

Anisotropic Fuzz | Ambient Lighting

GHOST

- Want anisotropy to continue to be visible in ambient lighting conditions
  - For us this means spherical harmonic lighting + specular probes
- Consider the rendering equation for our BRDF:

Speaker notes

We want anisotropy to continue to be visible under indirect lighting;
[next]
for us this means spherical harmonic lighting + reflection probes.
[next]
The rendering equation for our BRDF looks like this, with the terms in the blue box being proportional to the isotropic scattering diagram we saw earlier.

- Want anisotropy to continue to be visible in ambient lighting conditions
  - For us this means spherical harmonic lighting + specular probes
- Consider the rendering equation for our BRDF:

$$L_o(\hat{\mathbf{v}}) = \mathbf{c}_f \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}}) g(\hat{\mathbf{u}}, \hat{\mathbf{v}}) L_i(\hat{\mathbf{u}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

- Approximate using:

$$L_o(\hat{\mathbf{v}}) \approx \frac{\mathbf{c}_f}{2\pi} \int_\Omega g(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}}) L_i(\hat{\mathbf{u}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}},$$

Speaker notes

Let's call that g; recall that it was approximately constant over the hemisphere,
[next]
so we're going to approximate the integral
[next]
by pulling g out and multiplying by its
[next]
average value over the hemisphere; that is, the integral of g over the hemisphere divided by two pi.

$$L_o(\hat{\mathbf{v}}) \approx \frac{\mathbf{c}_f}{2\pi} \int_\Omega g(\hat{\mathbf{u}}, \hat{\mathbf{v}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}}) L_i(\hat{\mathbf{u}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

- Now assume that L_i is approximately constant

$$L_o(\hat{\mathbf{v}}) \approx \frac{\mathbf{c}_f}{4\pi^2} \int_\Omega g(\hat{\mathbf{u}}, \hat{\mathbf{v}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega L_i(\hat{\mathbf{u}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

Speaker notes

[next]
We do the same thing again, assuming L_i is approximately constant,
[next]
so we pull it out of the integral,
[next]
and again multiply by its average value over the hemisphere.

- We've broken it down into:

$$L_o(\hat{\mathbf{v}}) \approx \mathbf{c}_f$$

$$\times \frac{1}{2\pi} \int_\Omega g(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \approx \frac{d(1+c_0 \hat{\mathbf{v}}\cdot\hat{\mathbf{n}}+c_1(\hat{\mathbf{v}}\cdot\hat{\mathbf{n}})^2)}{d+c_2 \hat{\mathbf{v}}\cdot\hat{\mathbf{n}}+c_3(\hat{\mathbf{v}}\cdot\hat{\mathbf{n}})^3} \equiv \widetilde{G}(d, \hat{\mathbf{v}})$$

$$\times \frac{1}{2\pi} \int_\Omega L_i(\hat{\mathbf{u}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$ Easy to compute from SH coefficients

$$\times \int_\Omega p(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} = 1 - P_s(\hat{\mathbf{v}})$$

Speaker notes

So we're left with three integrals;
[next]
The first, the integral of g over the hemisphere, is well approximated by this expression, which we call G~, with the constants again determined using curve fitting;
[next]
the second integral, the average of the indirect light over the hemisphere, is easy to compute from SH coefficients;
[next]
this last term, the integral of the phase function over the hemisphere, is one minus the P_s function that we saw earlier, evaluated this time in the view direction. This term is responsible for the anisotropic response.

[Additional Notes]
Please see the Supplemental Material for details regarding the approximate form of the first integral.

For the second integral, if we start with the SH coefficients for the direct evaluation of (cosine-weighted) irradiance as in [RH01a], the SH coefficients that approximate the average luminance over the hemisphere are obtained by multiplying the first three SH bands by 1/pi, 3/(4*pi), and 0 respectively.

Anisotropic Fuzz | Ambient Shadowing

- Also need to account for ambient shadowing of base layer
- First consider diffuse ambient lighting only
- Define:
  - E_p: Irradiance of the light penetrating the scattering layer
  - E_s: Irradiance of the light scattered down by the scattering layer
  - $\mathbf{c}_b$: Diffuse albedo of the base layer
  - $L_{b,o}$: Luminance of the base layer due to diffuse ambient lighting

$$L_{b,o}(\hat{\mathbf{v}}) = \frac{\mathbf{c}_b}{\pi} P_e(\hat{\mathbf{v}}) \left(E_p + E_s\right)$$

$$E_p = \int_\Omega P_p(\hat{\mathbf{u}}) L_i(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

$$E_s = \mathbf{c}_f \int_\Omega \left[1 - P_p(\hat{\mathbf{u}})\right] P_s(\hat{\mathbf{u}}) L_i(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

Bonus Slide

Speaker notes

Again, to conserve energy, we need to account for ambient shadowing of the base layer by the scattering layer. We'll start by consider ambient diffuse light only.
[next] Define the following terms:
E_p is the irradiance of the light that penetrates through the scattering layer to the base layer
E_s is the irradiance of the light that is scattered towards the base layer by the scattering layer
c_b is the diffuse albedo of the base layer
L_{d,o} is the luminance of the base layer as seen along the view vector, due to ambient diffuse light
[next]
L_{d,o} is given by this equation,
[next]
that is, the product of P_e, the probability of escaping through the scattering layer along v,
[next] times the Lambertian BRDF, [next] times the sum of the irradiance values E_p [next] and E_s.
[next]
E_p is given by this equation,
[next]
that is, the integral over the hemisphere of the product of P_p, the probability of penetrating the scattering layer,
[next] times the luminance in direction u, [next] times the cosine term.
[next] Similarly, E_s is given by this equation, [next] which is the fiber albedo
[next] times the integral over the hemisphere of the product of 1 - P_p, the probability of being scattered by the scattering layer,
[next] times the probability of being scattered towards the base layer, [next] times the luminance in direction u,
[next] times the cosine term.

$$E_p = \int_\Omega P_p(\hat{\mathbf{u}}) L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

- Approximate using:

$$E_p \approx \frac{1}{\pi} \int_\Omega P_p(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

  - Exact for uniform lighting

Bonus Slide

Speaker notes

First we'll consider the expression for E_p, the irradiance penetrating the scattering layer.
[next]
We'll approximate by assuming that L_i is constant,
[next]
and separating the integral into
[next]
the product of the cosine-weighted average of P_p over the hemisphere,
[next]
and the irradiance on the surface.

- We've broken it down into:

$$E_p \approx$$

$$\frac{1}{\pi}\int_\Omega P_p(\hat{\mathbf{u}})(\hat{\mathbf{u}}\cdot\hat{\mathbf{n}})\,\mathrm{d}\omega_{\hat{\mathbf{u}}} \qquad \approx 1 + c_0 d + c_1 d^2 \equiv \widetilde{Q}(d)$$

$$\times \int_\Omega L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}}\cdot\hat{\mathbf{n}})\,\mathrm{d}\omega_{\hat{\mathbf{u}}} \qquad \equiv E\text{, the irradiance on the surface (computed using SH)}$$

Bonus Slide

Speaker notes

[next]
We used curve fitting to approximate the first integral, which we call Q~,
[next]
while the second integral is the simply E, the irradiance on the surface, which can be computed from standard SH coefficients, as in [RH01a].

[Additional Notes]
Please see the Supplemental Material for details regarding the approximate form of the first integral.

- Now for irradiance of light scattered onto base layer by scattering layer:

$$E_s = \mathbf{c}_f \int_\Omega \left[1 - P_p(\hat{\mathbf{u}})\right] P_s(\hat{\mathbf{u}}) L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

- Approximate using:

$$E_s \approx \frac{\mathbf{c}_f}{\pi^2} \int_\Omega \left[1 - P_p(\hat{\mathbf{u}})\right] (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega P_s(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}} \int_\Omega L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

  - Exact for isotropic scattering and uniform lighting

Bonus Slide

Speaker notes

We'll now consider E_s, the irradiance on the base layer due to scattering.
[next]
Approximate it using this equation, which is the fiber color times
[next]
the cosine-weighted average of the probability of being scattered,
[next]
times the cosine-weighted average of the probability of being scattered towards the base layer,
[next]
times the irradiance on the surface.
[next]
Note that this is exact for isotropic scattering and uniform lighting.

- We've broken it down into:

$$E_s \approx \mathbf{c}_f$$

$$\times \frac{1}{\pi} \int_\Omega \left[1 - P_p(\hat{\mathbf{u}})\right] (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \qquad = 1 - \frac{1}{\pi} \int_\Omega P_p(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

$$\times \int_\Omega P_s(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \qquad \equiv R(\theta_f, \alpha), \text{ constant for fixed } \theta_f \text{ and } \alpha$$

$$\times \int_\Omega L_i(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \qquad \equiv E, \text{ the irradiance on the surface (computed using SH)}$$

Bonus Slide

Speaker notes

[next]
The first integral is simply one minus Q~, the expression we developed two slides earlier for the first integral in the approximation of E_p;
[next]
the second integral is constant for constant spread (i.e., Schlick k or GGX alpha) and fiber tilt, so it can be evaluated numerically;
[next]
while the last integral is again the irradiance on the surface, which we also saw two slides ago.

We now have all the pieces we need to compute the luminance of the base layer due to ambient diffuse lighting, taking shadowing by the scattering layer into account.

[Additional Notes]
Please see the Supplemental Material for details on how we calculate the second integral.

- Putting it all together, we have

$$
\begin{aligned}
L_{b,o}(\hat{\mathbf{v}}) &= \frac{\mathbf{c}_b}{\pi} P_e(\hat{\mathbf{v}}) \left(E_p + E_s\right) \\
&\approx \frac{\mathbf{c}_b}{\pi} P_e(\hat{\mathbf{v}}) \left[\widetilde{Q} E + \mathbf{c}_f \left(1 - \widetilde{Q}\right) R E\right] \\
&= \frac{\mathbf{c}_b \mathbf{c}_{\text{atten}}}{\pi} E
\end{aligned}
$$

where

$$
\mathbf{c}_{\text{atten}} = P_e(\hat{\mathbf{v}}) \left[\widetilde{Q} + \mathbf{c}_f R \left(1 - \widetilde{Q}\right)\right]
$$

Bonus Slide

Speaker notes

Combining our approximations for E_p and E_s into the equation for L_{b,o} lets us derive an expression c_atten for the attenuation of diffuse ambient lighting by the scattering layer, given here.

- For specular ambient shadowing, we follow a similar approach
  - Replacing SH sample with e.g. filtered reflection probe sample

$$\int_{\Omega} L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}} \rightarrow \int_{\Omega} f_{spec}(\hat{\mathbf{u}}, \hat{\mathbf{v}}) L_i(\hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})\, \mathrm{d}\omega_{\hat{\mathbf{u}}}$$

  - This lets us use the same $\mathbf{c}_{\mathrm{atten}}$ attenuation term

Bonus Slide

Speaker notes

For simplicity, we treat specular ambient shadowing in a similar manner to diffuse ambient shadowing,
[next]
replacing the SH samples for evaluating irradiance with the filtered reflection probe sample.
[next]
Therefore we use the same attenuation term derived for diffuse ambient lighting, which avoids additional runtime cost.

Anisotropic Fuzz | Simplified Deferred Version

GHOST

- For more common environmental materials (such as moss), we implemented a simplified version of this BRDF for deferred shading
  - 7-bit fuzziness value in G-Buffer (mapped to [0, 1])
- Used a fixed set of parameters:
  - Density $d = 0.5$
  - Spread $= 0.9$ ($k_{\text{Schlick}} \approx -0.1$, or $\alpha_{\text{SGGX}} \approx 0.95$)
  - $\hat{\mathbf{t}}_{\text{Fiber}} = \hat{\mathbf{n}}_{\text{Vertex}}$
  - Fiber albedo $\mathbf{c}_{\text{Fiber}} = 5 \cdot \mathbf{c}_{\text{Lambert}}$
- For performance reasons, did not include base layer shadowing; instead we used:

$$L_{\text{Diffuse}} = \text{lerp}\left(L_{\text{Lambert}}, L_{\text{Fuzz}}, \text{fuzziness}\right)$$

Speaker notes

The world of Tsushima is green and lush, and features a lot of moss-covered rocks and buildings, as well as other fuzzy materials like the felt used for Mongol yurts, so Eric Wohllaib, one of our rendering programmers, implemented a simplified version of the BRDF for deferred shading.
[next]
It has a single parameter, a 7-bit fuzziness value;
[next]
all other parameters were set globally as follows:
[next]
The density is 0.5, which is fairly high;
[next]
the "spread" is 0.9, which is nearly uniform, and in hindsight we could have used uniform scattering and saved some instructions;
[next]
the fiber direction was set to the vertex normals (though due to the large spread this had little impact);
[next]
and the fiber albedo was fixed at five times the diffuse albedo.
[next]
For performance reasons, we didn't include base layer shadowing,
[next]
but instead we used a mixture of the Lambertian and fuzz BRDFs; since each conserves energy on its own, a mixture is also energy conserving.

Anisotropic Fuzz | Simplified Deferred Version

- In the forward-shaded version, we expose a parameter to smooth normals (towards vertex normals) to give the scattering layer a softer appearance
- Environment vertex normals were too inconsistent to use directly for shading
- Instead we used wrap lighting to soften normals (McAuley 2013):

$$\hat{\mathbf{u}} \cdot \hat{\mathbf{n}} \rightarrow \frac{1+n}{2(1+w)} \left(\frac{\max(\hat{\mathbf{u}} \cdot \hat{\mathbf{n}} + w, 0)}{1+w} \right)^n \text{ with } n = 2, w = 0.2$$

Speaker notes

In the forward shaded version, we smooth the appearance of the scattering layer by allowing the normals to be lerped towards the vertex normals.
[next]
For our environment art assets, vertex normals were too inconsistent to use directly for shading,
[next]
so instead we use energy-conserving wrap lighting
[next]
which helps to soften the appearance of the scattering layer.

[Additional Notes]
For some assets, smoothing the normals by blending with a normal computed by using a lower-resolution mipmap of the normal map gives better results than blending with vertex normals.

Anisotropic Fuzz | Results

- Ghost didn't have any crushed velvet-like materials that made full use of the BRDF's capabilities
- To demonstrate the technique, I created a “velvety” horse by modifying the Ghost horse shaders
- Note that these results use the SGGX phase function, which is different than what shipped in Ghost
  - In general, the results are qualitatively very similar
- Disclaimer: Programmer art ahead!

Speaker notes

I'll now show some results. In its final form, Ghost didn't have any crushed velvet-like materials that made full use of the BRDF's capabilities;
[next]
so to demonstrate I created a velvety horse by modifying the horse shaders.
[next]
Note that these results use the new SGGX phase, which is not what was shipped in Ghost,
[next]
though the results are qualitatively very similar.
[next]
Please excuse the programmer art!

Speaker notes

You can see here on the shadowed side that the anisotropic behavior of the material remains visible;
for reference this material uses a density of 0.2, a spread of 0.2 which corresponds to a GGX alpha of 0.53, a fiber polar angle of 23 degrees, and a noisy fiber direction map.

Speaker notes

Let me break that down; here we've got the base layer only with no scattering layer

Speaker notes

If we add the scattering layer with no base layer attenuation, it looks like this;

Speaker notes

Adding base layer attenuation (but while defining P_s, the fraction of scattered light assumed to be scattered towards the base layer, to be equal to 0), it looks like this; note that the material is darker, especially along the terminator visible on horse's neck.
[flip back and forth a few times]
Since P_s is 0, the shadowed side lacks any anisotropy;

[Additional Notes]
The original version of this slide incorrectly stated that P_s was set to 1 here.

Speaker notes

Implementing P_s gives the final BRDF, which is the same as what was shown in the video.

Speaker notes

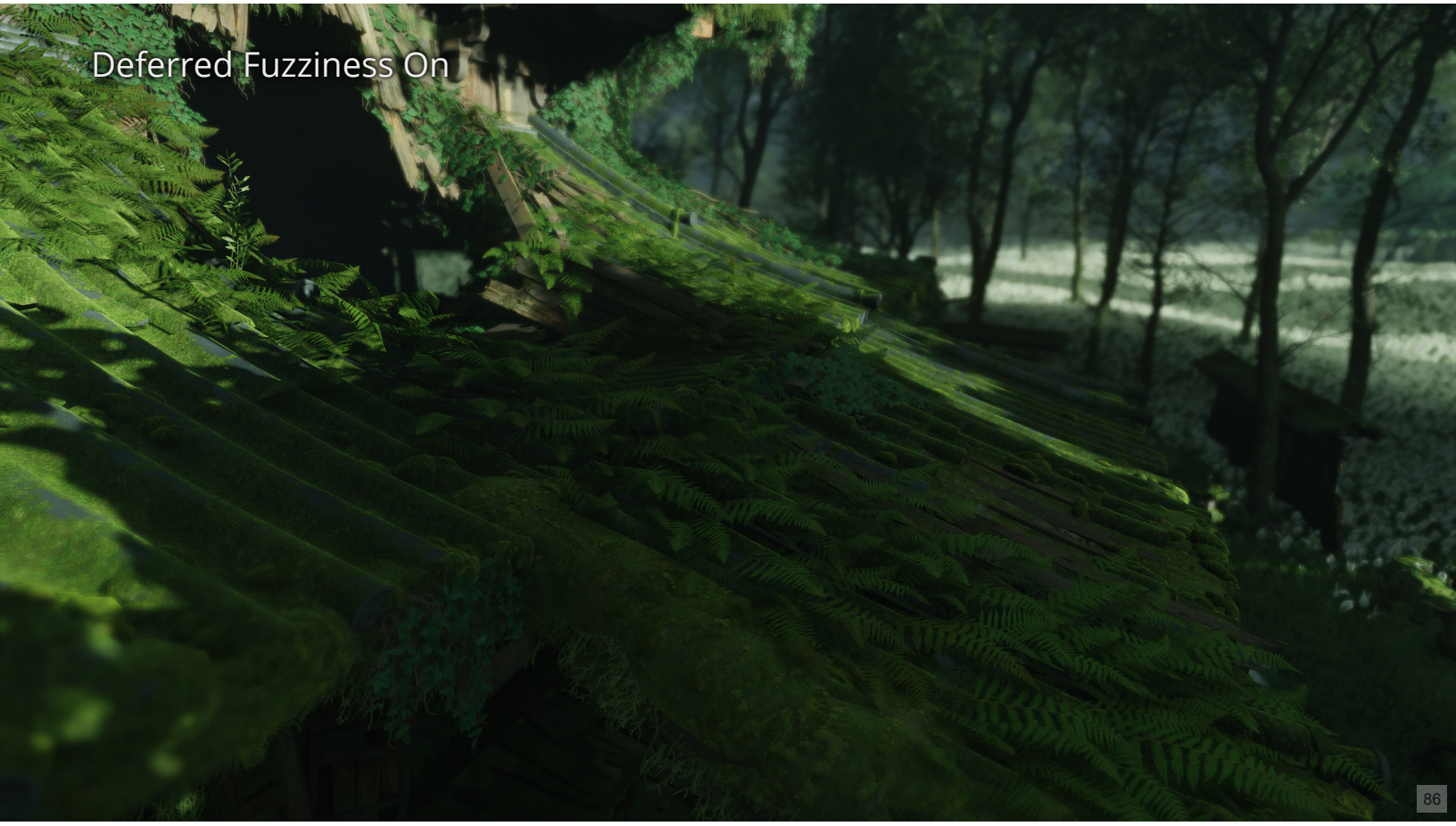
Here is the effect of the various model parameters; first we vary density;
[next]
then the fiber color value or luminance ;
[next]
the fiber color saturation;
[next]
the spread (which corresponds to GGX alpha)
[next]
and finally the fiber polar angle, where on the left the fibers point straight up, while on the right they are flat against the surface.

Speaker notes

Here's an example showing the deferred version; this is with it on,

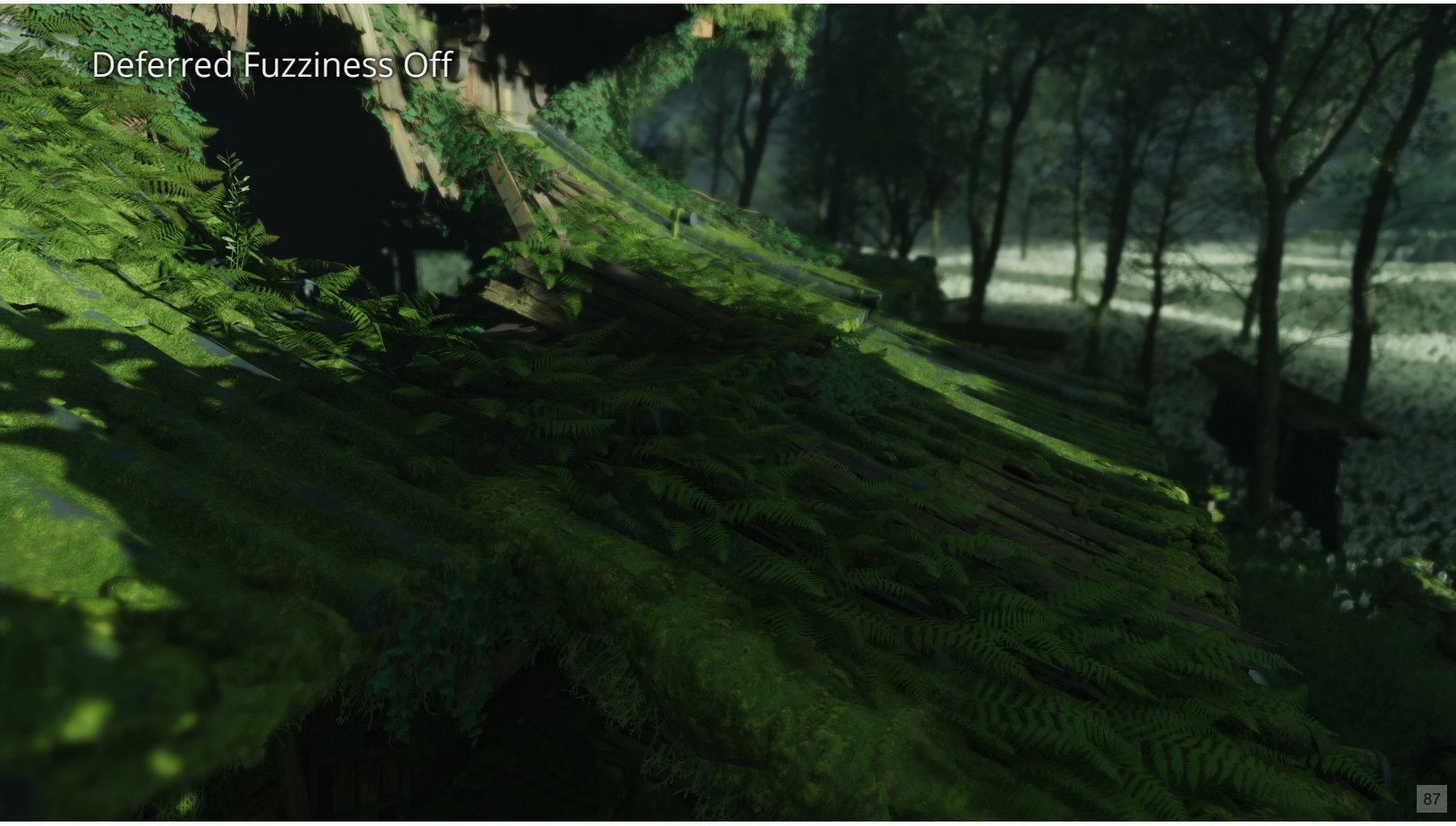
Speaker notes

and this is using Lambertian diffuse only.
[toggle back and forth a few times]



Speaker notes

Here's a close-up showing an area in shadow, which also retains the fuzzy appearance.



Speaker notes

versus Lambertian diffuse.
[toggle back and forth a few times].

Anisotropic Fuzz | Summary

- New anisotropic fuzz BRDF
  - Able to reproduce the appearance of materials like crushed velvet
    - Under direct and indirect light
  - Inexpensive enough to be used on PlayStation 4 hardware

Speaker notes

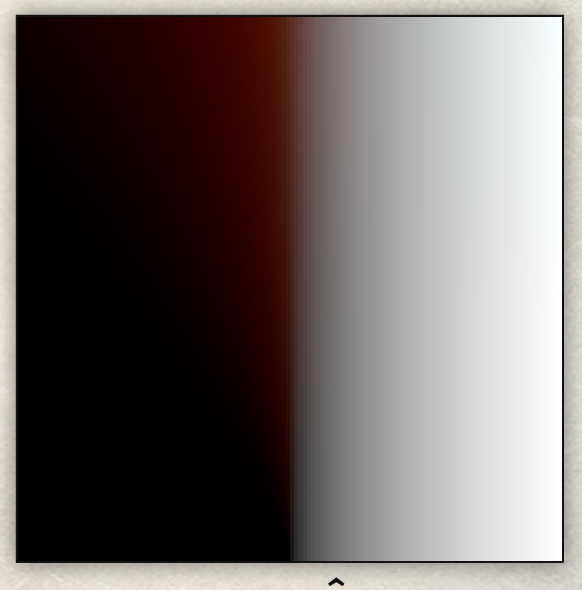
In summary, I've presented a new anisotropic fuzziness BRDF,
[next]
which is able to reproduce the appearance of materials like crushed velvet under direct and indirect light,
[next]
and is inexpensive enough to be used on PS4 hardware.

Speaker notes

Next, I'll discuss skin shading and the improvements we made there.

Skin Shading

- Using the *Pre-Integrated Skin Shading* technique (Penner and Borshukov, 2011)
- Uses LUTs to compute subsurface scattering based on $\hat{\mathbf{n}} \cdot \hat{\mathbf{l}}$ and curvature $\kappa = \frac{1}{r}$



$\frac{1}{r}$

$\hat{\mathbf{n}} \cdot \hat{\mathbf{l}}$

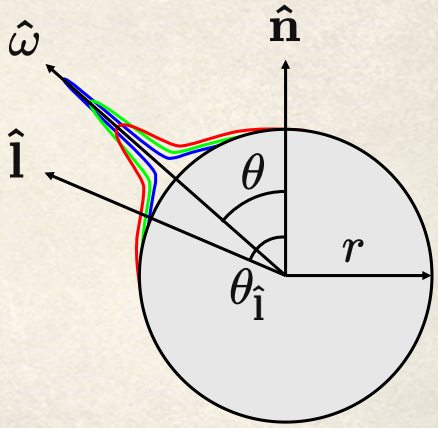
Speaker notes

We've been using pre-integrated skin shading since Infamous Second Son,
[next]
and it uses LUTs to compute subsurface scattering; for example, for punctual lights, it uses a LUT parameterized by the cosine of the angle between the light and the surface normal on one axis, and the curvature kappa (which is the reciprocal of the radius of curvature) on the other axis.

[Additional Notes]
We're using pre-integrated skin shading instead of a screen-space technique because it fit nicely into our existing lighting pipeline, and didn't require any extra G-buffer storage. Our skin shader also supports a "stubble" feature which also requires forward shading, so it made sense to incorporate pre-integrated skin shading into that pass. Our artists have also authored custom scattering profiles to simulate subsurface scattering in materials other than human skin, and this would be more difficult with a screen-space technique.

- The LUT is computed by performing integration on a “ring”:

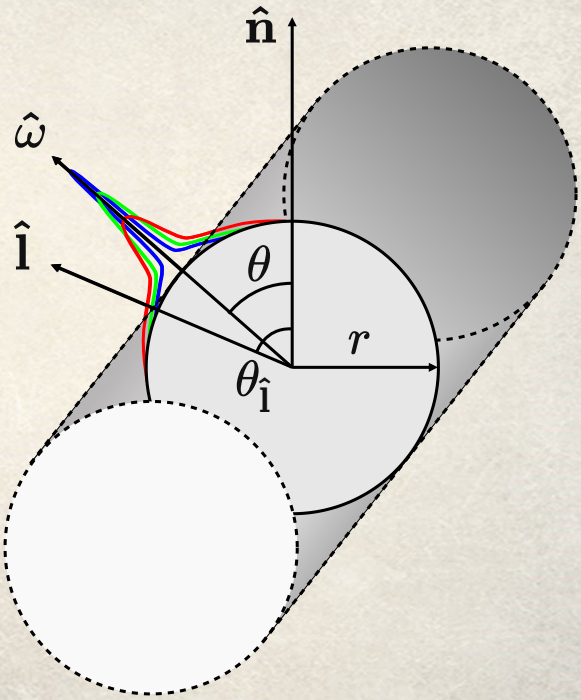
$$D\left(r, \cos\theta_{\hat{\mathbf{l}}}\right) = \frac{\int_{-\pi}^{\pi} \cos\left(\theta_{\hat{\mathbf{l}}} - \theta\right) R\left(2r \sin\left(\frac{\theta}{2}\right)\right) \mathrm{d}\theta}{\int_{-\pi}^{\pi} R\left(2r \sin\left(\frac{\theta}{2}\right)\right) \mathrm{d}\theta}$$



Speaker notes

It's worth revisiting the assumptions behind this LUT, because it's easy to get things wrong.
[next]
In the paper, the integration is performed on a "ring" as shown here, where we're integrating over theta for different curvatures and light angles,

- But it's better to think about it as being on a cylinder instead.
- Two things to notice:
  1. We want the curvature in the direction of the light
  2. We need to take care about which scattering profile we're using

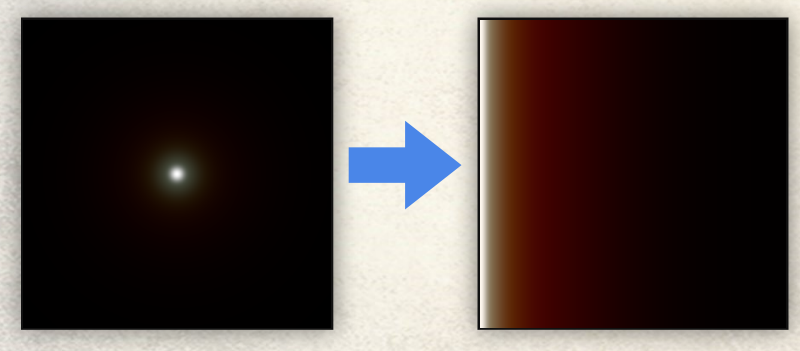


$$D\left(r, \cos\theta_{\hat{\mathbf{l}}}\right) = \frac{\int_{-\pi}^{\pi} \cos\left(\theta_{\hat{\mathbf{l}}} - \theta\right) R\left(2r\sin\left(\frac{\theta}{2}\right)\right) \mathrm{d}\theta}{\int_{-\pi}^{\pi} R\left(2r\sin\left(\frac{\theta}{2}\right)\right) \mathrm{d}\theta}$$

Speaker notes

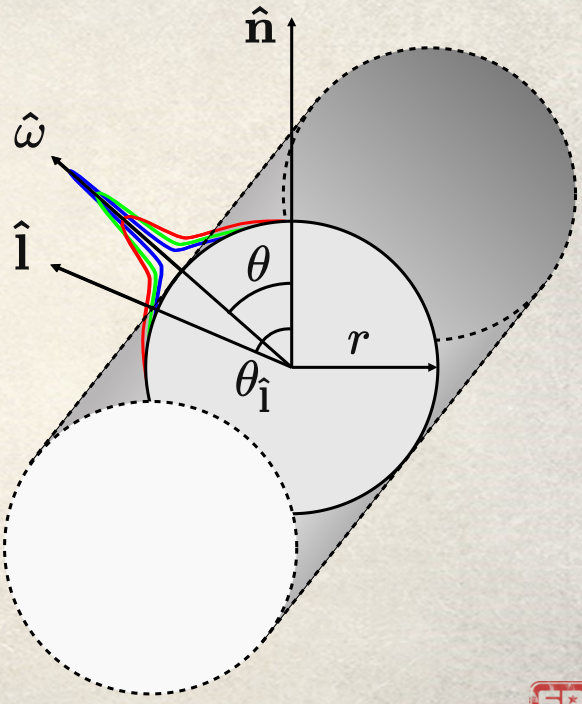
but it's better to think about it as being performed on a cylinder instead, which has a 2D surface like the skin that we're modelling.
[next]
Looking at it this way, we noticed two things;
[next]
The first is that we want the curvature in the directional of the light, and not for example the mean curvature;
[next]
and we also need to be careful about which scattering profile we're using.

$$D\left(r, \cos\theta_{\hat{\mathbf{l}}}\right) = \frac{\int_{-\pi}^{\pi} \cos\left(\theta_{\hat{\mathbf{l}}} - \theta\right) R\left(2r\sin\left(\frac{\theta}{2}\right)\right) \mathrm{d}\theta}{\int_{-\pi}^{\pi} R\left(2r\sin\left(\frac{\theta}{2}\right)\right) \mathrm{d}\theta}$$



radial

linear



Speaker notes

[next]
The scattering profile in this equation is R;
[next]
In the literature you typically see radial profiles, obtained by shining a bright light on a point and measuring the scattered light.
[next]
The other type of profile you could use is a linear profile, which is what you would get if you used an infinite linear light source.
[next]
Because of the symmetry along the cylindrical axis, when integrating on a ring you should use the linear profile,
[next]
Which can be obtained by pre-integrating the radical profile.

[Additional Notes]
Note that when integrating against SH zonal harmonics. you want to use the radial profile, and integrate over the sphere -- cylinders are not radically symmetric!

- We want the *directional* curvature, not just the mean curvature
- Can calculate directional curvature using the curvature tensor

$$\mathbb{II} = \left[\hat{\mathbf{d}}_1, \hat{\mathbf{d}}_2\right] \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \left[\hat{\mathbf{d}}_1, \hat{\mathbf{d}}_2\right]^T$$

  - $\hat{\mathbf{d}}_\mathbf{i}$: Principal directions
  - κ_i: Principal curvatures
- The curvature in direction $\hat{\mathbf{l}}$ is found by

$$\kappa_{\hat{\mathbf{l}}} = \hat{\mathbf{l}}^T \mathbb{II} \hat{\mathbf{l}}$$

Speaker notes

The other thing that we mentioned was that we want the directional curvature, not just the mean curvature.
[next]
We can calculate directional curvature using the curvature tensor, also called the second fundamental form, or just "two". d1 and d2 are the principal directions, while kappa1 and kappa2 are the corresponding principal curvatures.
[next]
The curvature in a particular direction l ("ell") is given by this simple formula.

- The curvature tensor $\mathbb{II}$ is a symmetric 2x2 matrix
- Store $\mathbb{II}$ + mean curvature in 4-byte vertex channel
  - Ambient lighting uses mean curvature
- Calculate tangent-space curvature tensors as a pre-process
  - Used algorithm described in *Estimating Curvature and Their Derivatives on Triangle Meshes* (Rusinkiewicz, 2004)
- Blurred resulting curvature slightly
  - Smooths out very high curvature at isolated vertices
  - Used an equal blend of two Gaussians ($\sigma_1 = 0.8\ \mathrm{cm}, \sigma_2 = 0.23\ \mathrm{cm}$)
  - Clamped concave curvature to zero for blur

Speaker notes

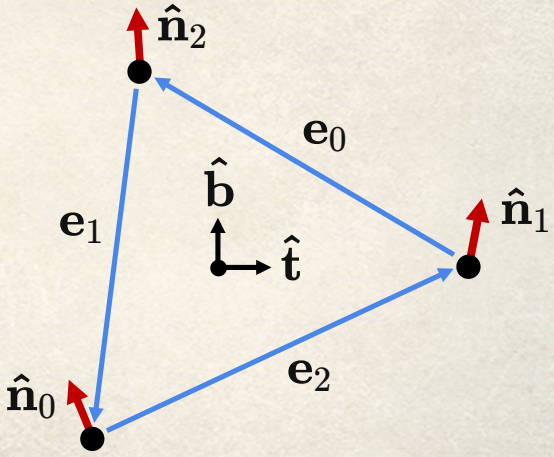
Since the curvature tensor is a symmetric 2x2 matrix,
[next]
we can store it plus the mean curvature, which we use for ambient lighting, in a 4-byte vertex channel.
[next]
We calculate the tangent-space curvature tensors as a pre-process, using the algorithm described by Rusinkiewicz in this paper from 2004.
[next]
We found it necessary to blur the resulting curvature tensors, to smooth out very high curvature at isolate vertices such as at the corners of the mouth or around the eyes. We blurred on scale similar to the scale on which subsurface scattering occurs, which gave good results. When blurring curvature here it's important to clamp negative concave curvature to zero, so that concave areas don't "cancel out" nearby convex curvature.

- Calculate $\mathbb{II}$ for each face using least squares with the following constraints:

$$\mathbb{II}\begin{bmatrix}\mathbf{e}_0\cdot\hat{\mathbf{t}}\\ \mathbf{e}_0\cdot\hat{\mathbf{b}}\end{bmatrix}=\begin{bmatrix}(\hat{\mathbf{n}}_2-\hat{\mathbf{n}}_1)\cdot\hat{\mathbf{t}}\\ (\hat{\mathbf{n}}_2-\hat{\mathbf{n}}_1)\cdot\hat{\mathbf{b}}\end{bmatrix}$$

$$\mathbb{II}\begin{bmatrix}\mathbf{e}_1\cdot\hat{\mathbf{t}}\\ \mathbf{e}_1\cdot\hat{\mathbf{b}}\end{bmatrix}=\begin{bmatrix}(\hat{\mathbf{n}}_0-\hat{\mathbf{n}}_2)\cdot\hat{\mathbf{t}}\\ (\hat{\mathbf{n}}_0-\hat{\mathbf{n}}_2)\cdot\hat{\mathbf{b}}\end{bmatrix}$$

$$\mathbb{II}\begin{bmatrix}\mathbf{e}_2\cdot\hat{\mathbf{t}}\\ \mathbf{e}_2\cdot\hat{\mathbf{b}}\end{bmatrix}=\begin{bmatrix}(\hat{\mathbf{n}}_1-\hat{\mathbf{n}}_0)\cdot\hat{\mathbf{t}}\\ (\hat{\mathbf{n}}_1-\hat{\mathbf{n}}_0)\cdot\hat{\mathbf{b}}\end{bmatrix}$$



- Combine face $\mathbb{II}$ on vertices using “Voronoi area” weighting
  - See paper for details

Speaker notes

To calculate the curvature tensors for a mesh, we start by calculating the curvature tensors for each face,
[next]
which is done using least-squares with the following constraints. The unknowns are the three components of the symmetric 2x2 curvature tensor on the left.
[next]
Once we have those we combine the face tensors onto vertices; I'll refer you to the paper for the details.

Skin Shading | Results

- Comparison of:
  - Zero curvature
  - Mean curvature
  - Directional curvature

Speaker notes

I'll now show some comparisons between
[next]
no curvature,
[next]
mean curvature,
[next]
and directional curvature.

Speaker notes

This is the result with a curvature of zero everywhere,

Speaker notes

Here is the result with mean curvature

Speaker notes

and this is the result when using directional curvature.

Speaker notes

Since the effect is hard to see there, I'll show the same things again with ambient lighting off and exposure and tonemapping adjusted to make the scattering more visible. Again, this is no curvature,

Speaker notes

Mean curvature,

Speaker notes

And directional curvature.

Speaker notes

At the top we have the respective curvature values that are used for shading; I'll point out a few of the differences here.
[next]
On the bridge of the nose, there is more scattering when using directional curvature because the light is coming from the side, and the curvature is highest in that direction.
[next]
Whereas there is less scattering on the nostrils, because the curvature is primarily perpendicular to the light direction. Using directional scattering reduces the unnatural scattering that we see when using mean curvature.
[next]
Similarly, the lips also have very long-range scattering when using mean curvature, which is reduced when using directional curvature.

Speaker notes

Here is another example with the light pointing almost straight down;
[next]
Here we see very long-range scattering along the cheeks when using mean curvature,
[next]
which is reduced to a more realistic range when using directional curvature.

```
float CurvatureFromLight(
    float3 tangent,
    float3 bitangent,
    float3 curvTensor,
    float3 lightDir)
{
    // Project light vector into tangent plane

    float2 lightDirProj = float2(dot(lightDir, tangent), dot(lightDir, bitangent));

    // NOTE (jasminp) We should normalize lightDirProj here in order to correctly
    //    calculate curvature in the light direction projected to the tangent plane.
    //    However, it makes no perceptible difference, since the skin LUT does not vary
    //    much with curvature when N.L is large.

    float curvature = curvTensor.x * GSquare(lightDirProj.x) +
                          2.0f * curvTensor.y * lightDirProj.x * lightDirProj.y +
                          curvTensor.z * GSquare(lightDirProj.y);

    return curvature;
}
```


Speaker notes

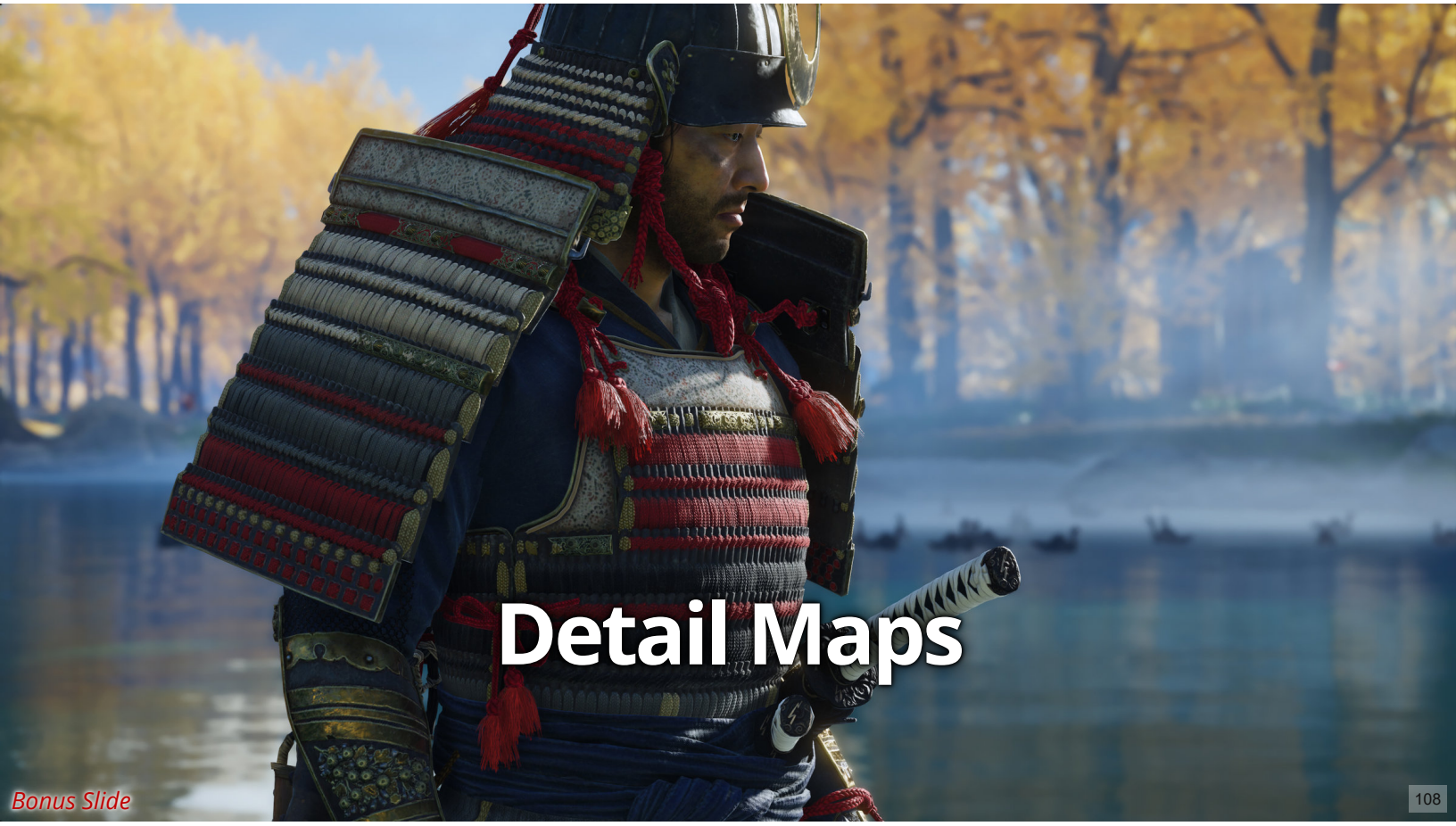
Here is some shader code showing how to calculate the directional curvature.
[next]
We've got a function that takes
[next]
the tangent,
[next]
bitangent,
[next]
curvature tensor,
[next],
and light direction;
[next]
here we project the light direction into the tangent plane;
[next]
as an optimization we skipped the normalization of the projected light direction, since it made no perceptible difference in practice; this is because the skin LUT does not vary much when N dot L is large, and that is when the projected light direction, before normalization, is shortest.
[next]
This is the multiplication of the light direction with the curvature tensor,
[next]
which yields the directional curvature.

Skin Shading | Summary

- New way of thinking about pre-integrated skin shading
  - Use linear scattering profile/**cylindrical** integration for punctual light LUT generation
  - Radial profile with **spherical** integration for SH lighting LUT
  - Importance of directional curvature for improved accuracy

Speaker notes

In summary, I've presented a new way to think about pre-integrated skin shading,
[next]
which emphasizes the importance of using a linear scattering profile or cylindrical integration when computing the punctual light LUT.
[next]
I'll also mention here that it's also important to use the radial profile together with *spherical* integration when computing the SH lighting LUT.
[next]
I've also demonstrated the importance of using directional curvature, which improves the quality of the subsurface scattering computed by this technique.



Speaker notes

In this section, I'll discuss our approach to authoring character detail maps in Ghost of Tsushima.

Detail Maps

- Goals:
  - Reduce texture memory usage by separating low- and high-frequency content
  - Use library of tileable physically based materials for high frequency
  - Intuitive authoring of low frequency (wear, grime, etc.)
  - Combine in an efficient way at runtime
- For normal maps, *Reoriented Normal Mapping* (Barré-Brisebois and Hill, 2012) works well
- Want something that works well for other maps too (albedo, specular, gloss, AO, etc.)

Bonus Slide

Speaker notes

[next]
Our primary goal when using detail maps is to retain high levels of texture detail while reducing memory usage.
[next]
We also wanted to be able to use a library of tileable physically based materials as the basis for the high-frequency content,
[next]
while allowing intuitive authoring of low frequency content such as wear, grime, etc.,
[next]
and we needed to able to combine these two components efficiently at runtime.
[next]
Reoriented Normal Mapping solves this problem for normal maps,
[next]
but we wanted something that worked well for other types of maps as well.

Detail Maps | Traditional Approach

- Traditionally, tiled detail maps are used to add high-frequency detail on top of low-frequency maps
  - Using some kind of blending operator (overlay, additive, etc.)
- No guarantee that the result makes sense
  - Physically
  - Matching desired appearance
  - Needs interactive tuning
- Unclear how to generate from scanned materials
  - Usually needs preprocessing
- Combining two compressed textures
  - Potential for increased artifacts

Bonus Slide

Speaker notes

Traditionally, a detail map workflow starts with authored low-frequency content and adds tiled high-frequency detail
[next]
For many types of maps, there's no guarantee that the result makes sense physically, and it usually requires interactive tuning to match the desired appearance.
[next]
It was unclear to us how best to generate these maps from scanned materials, since detail maps usually need preprocesssing.
[next]
Finally, we realized that the traditional workflow combines two independently compressed textures, so could result in increased compression artifacts.

Detail Maps | Our Approach

- Start with tiled materials and paint low-frequency content on top
- Synthesize a map such that:

$$\text{Synthesized Map} \otimes \text{Tiled Detail Layers} = \text{Target Map}$$

$$s \otimes d = t$$

Bonus Slide

Speaker notes

Instead, we decided to start with our tiled materials and paint low-frequency content on top to create a "target" map.
[next]
We then synthesize a map such that
[next]
blending the synthesized map with the tiled detail layers matches the authored target map (as closely as possible).
[next]
In the following slides, the synthesized map will be s, the tiled detail map(s) will be d, and the target map will be t.

Detail Maps | Workflow

- Use Substance Designer and Substance Painter for authoring
  - Added SD nodes that match our shader UV and HSV transforms
- Set up tiled materials (up to 3 combined with masks)
- Paint on top to generate a target map
  - This is what artists export for use in our shaders
- At asset compile time:
  - Process target + tiled layers and masks
  - Generate a new synthesized map

Bonus Slide

Speaker notes

Artists use Substance Designer and Painter to author target maps; this required adding Designer nodes that match our shader UV and HSV transforms.
[next]
Artists first set up the tiled materials (we supported up to 3, combined with masks),
[next]
and they then paint on top of these to generate a target map,
[next]
which is exported for use in our shaders.
[next]
At asset compile time, we read the target map along with the tiled layers and masks, and generate a new synthesized map by inverting the blending operator.

- What blend operator $\otimes$ should we use?
- Need to ensure that a solution exists for s in

$$s \otimes d = t \quad \forall d, t \in [0, 1]$$

  - s: Synthesized map
  - d: Detail map
  - t: Authored target map
- Excludes hard light, soft light, screen, multiply, etc.
- Overlay works fairly well:

$$f_{overlay}(s, d) = \begin{cases} 2sd, & s < 0.5 \\ 1 - 2(1 - s)(1 - d), & s \geq 0.5 \end{cases}$$

Bonus Slide

Speaker notes

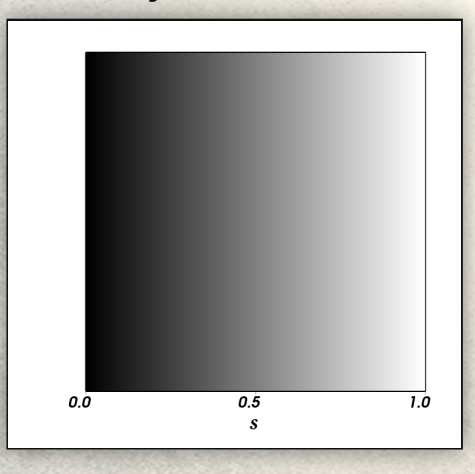
We needed to choose a blending operator,
[next]
such that a solution exists for s in the blending equation for all values of d and t.
[next]
This excludes a number of operators,
[next]
but overlay blending, given by this equation, works reasonably well.

[Additional Notes]
Overlay has the nice property that if either of s or d are 0.5, then the other value is passed through unmodified.

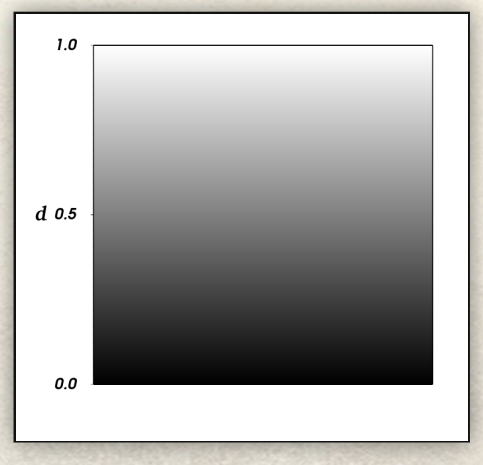
Detail Maps | Blending

$$f_{overlay}(s,d) = \begin{cases} 2sd, & s < 0.5 \\ 1 - 2(1-s)(1-d), & s \geq 0.5 \end{cases}$$

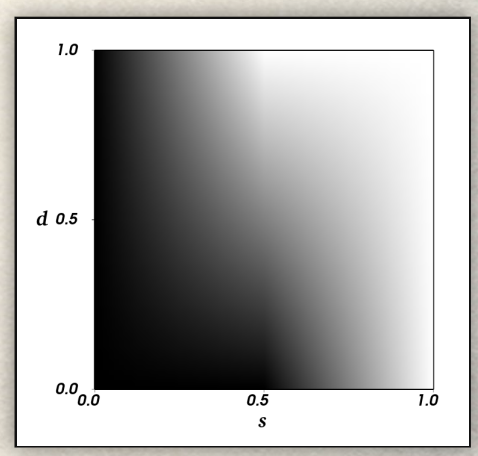
Synthesized



Detail



Target

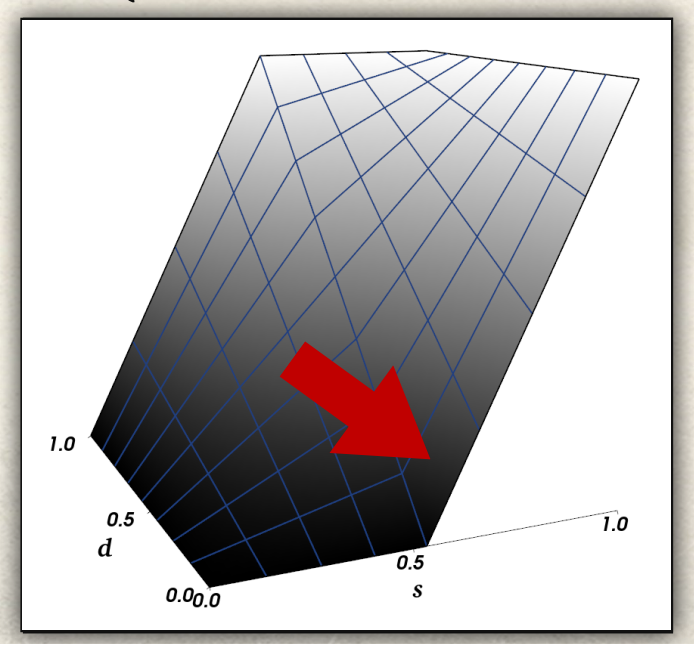


Bonus Slide

Speaker notes

[next]
For a synthesized gradient from left to right,
[next]
blended with
[next]
a detail gradient from bottom to top,
[next]
the result of overlay blending
[next]
is this.

$$f_{overlay}(s,d) = \begin{cases} 2sd, & s < 0.5 \\ 1 - 2(1-s)(1-d), & s \geq 0.5 \end{cases}$$



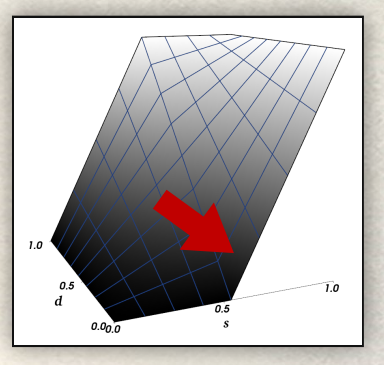
Bonus Slide

Speaker notes

[next]
Looking down the d (high frequency detail) axis, we see that the whole range of output values is attainable by some value of s for any value of d, which is what we need.
[next]
The opposite is not true -- the effect of d is reduced as we approach 0 or 1 (as can be seen by the decreasing slopes).
[next]
In 3D, it looks like this.
[next]
Note the part of the curve pointed out by the arrow, which corresponds to dark detail d being brightened by the low frequency synthesized map s.

Detail Maps | Precision

- Precision gets worse as detail d approaches 0 and 1
  - Relative precision especially bad as d approaches 0
- Noticeable quality degradation when using BC1
- Wanted to increase precision when target t is near d
  - Especially when brightening dark tiled layers
  - Led us to modify the blending equation:



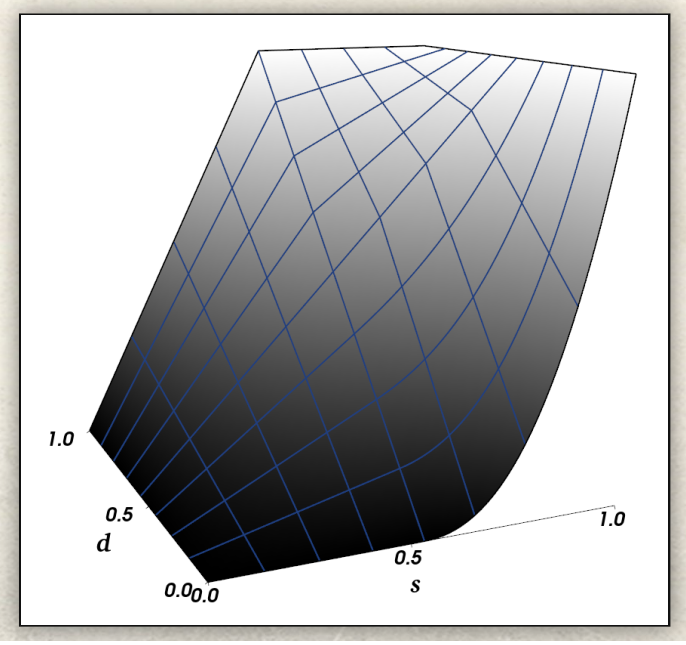
$$f_{os}(s,d) = \begin{cases} 2sd, & s < 0.5 \\ 1 - 2(1-s)(1-d), & s \geq 0.5 \text{ and } d \geq 0.5 \\ \text{lerp}\left(2sd, s, (2s-1)^2\right), & s \geq 0.5 \text{ and } d < 0.5 \end{cases}$$

Bonus Slide

Speaker notes

[next]
Due to the large gradient in the direction of the s axis when d approaches 0 or 1, precision here suffers, and relative precision is especially bad as d approaches 0 when s slightly larger than 0.5.
[next]
This led to noticeable quality degradation when using BC1 compression.
[next]
Under the assumption that the target and detail layers would usually be similar, we decided to increase the precision of the blending operator when t is near d, in particular when brightening dark tiled layers.
[next]
This led us to modify the blending operator as follows; we call this "smooth overlay".

$$f_{os}(s,d) = \begin{cases} 2sd, & s < 0.5 \\ 1 - 2(1-s)(1-d), & s \geq 0.5 \text{ and } d \geq 0.5 \\ \text{lerp}\left(2sd, s, (2s-1)^2\right), & s \geq 0.5 \text{ and } d < 0.5 \end{cases}$$

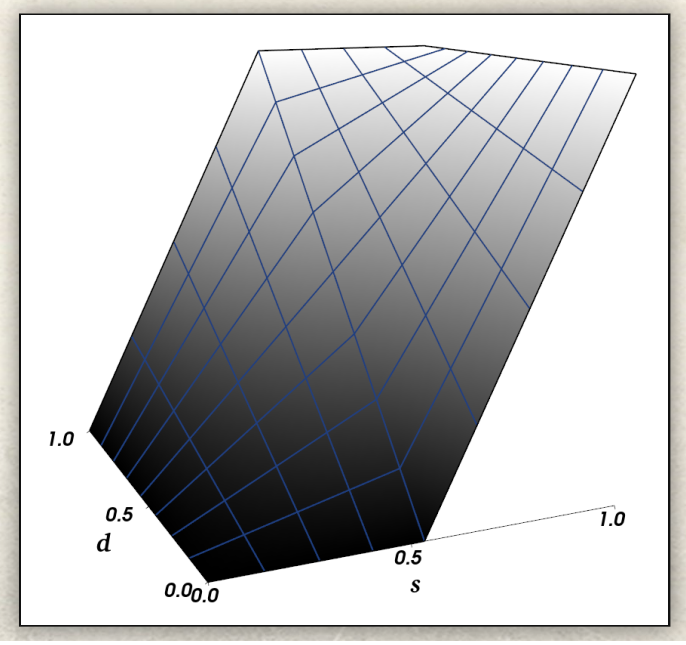


Bonus Slide

Speaker notes

[next]
The graph of the operator looks like this, looking down the t axis,
[next]
looking down the d axis,
[next]
looking down the s axis,
[next]
and in 3D.

$$f_{overlay}(s,d) = \begin{cases} 2sd, & s < 0.5 \\ 1 - 2(1-s)(1-d), & s \geq 0.5 \end{cases}$$



Bonus Slide

Speaker notes

Here is the standard overlay version again for comparison.

Detail Maps | Precision

Bonus Slide

Speaker notes

Let's look at an example.

Detail Maps | Precision

Bonus Slide

Speaker notes

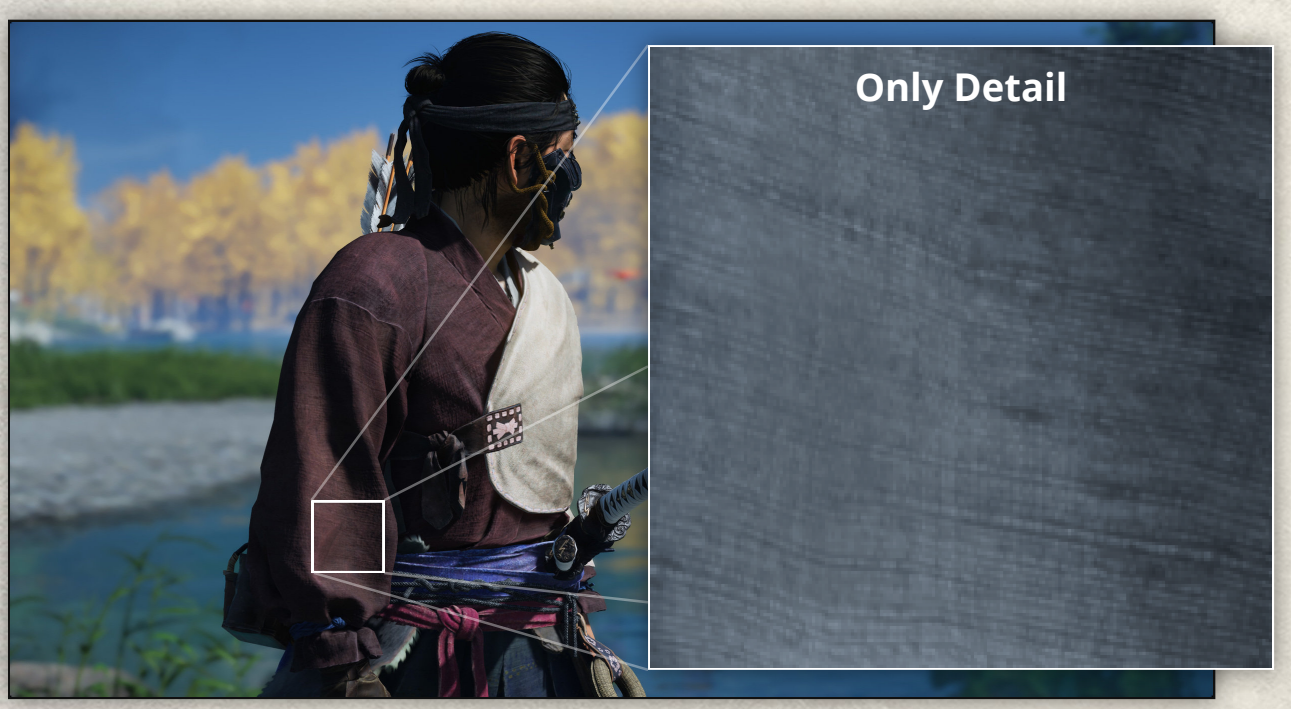
Here's a magnified look at a material in-game, which is a result of the blended synthesized and detail layers.

Detail Maps | Precision

Bonus Slide

Speaker notes

Without the detail layer it looks like this...

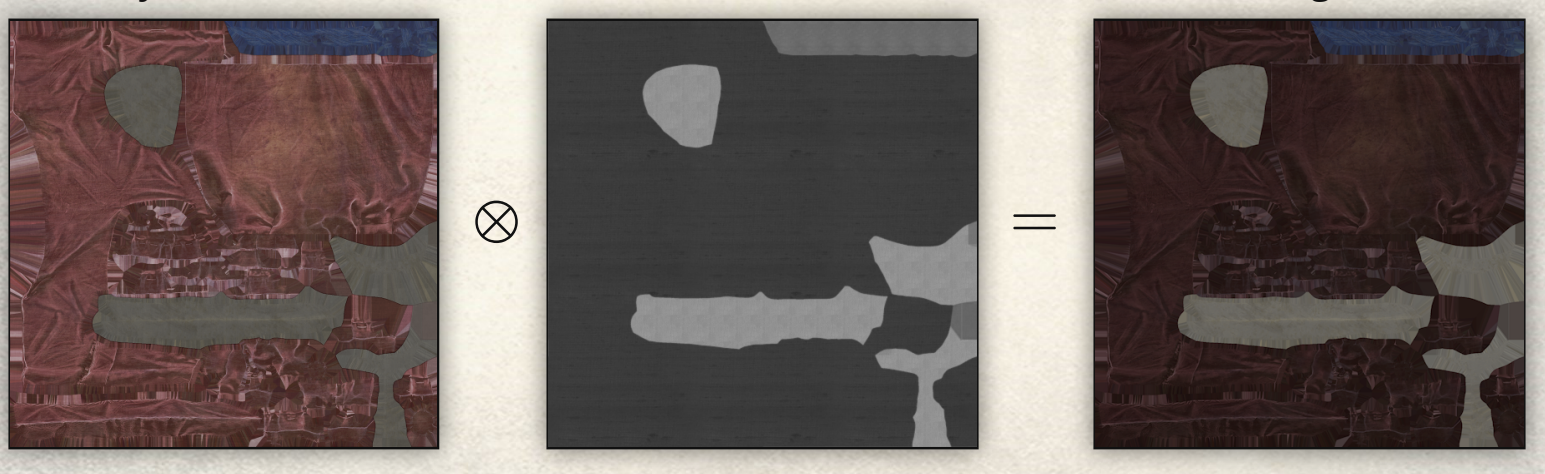


Bonus Slide

Speaker notes

... while it looks like this if we only look at the detail.

Detail Maps | Precision

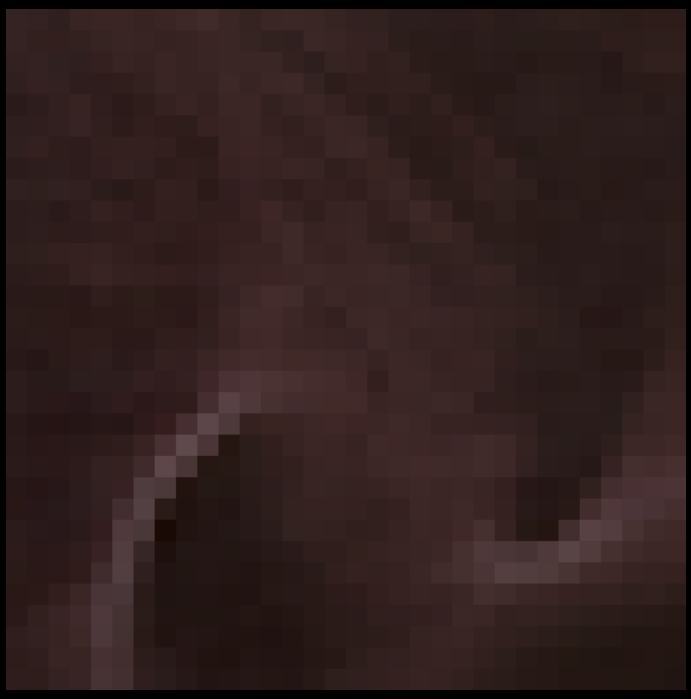


Bonus Slide

Speaker notes

These are the albedo maps for the same material; only the synthesized and detail maps are used at runtime.

Target



Bonus Slide

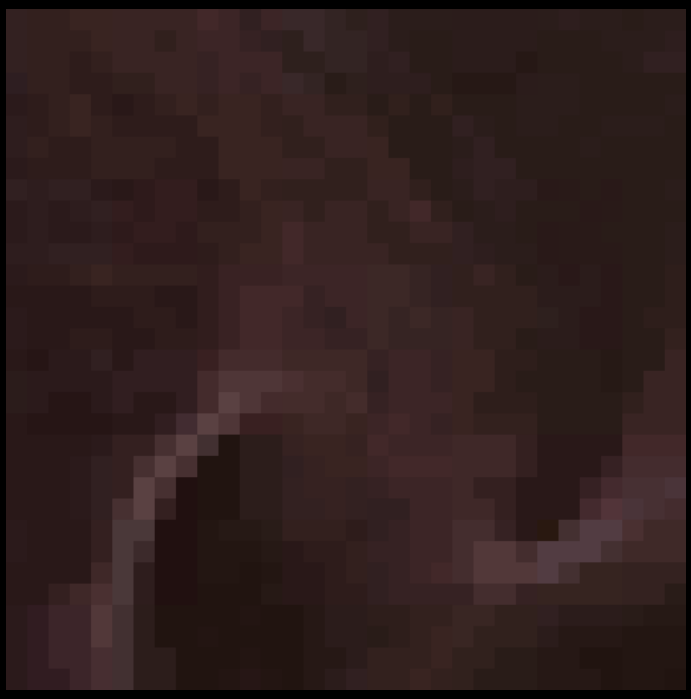
Speaker notes

This is the uncompressed target map, authored by the artists.
[next]
The zoomed area is shown by the white box in the left image.

Detail Maps | Precision

Target BC1

RMSE: 1.95



Bonus Slide

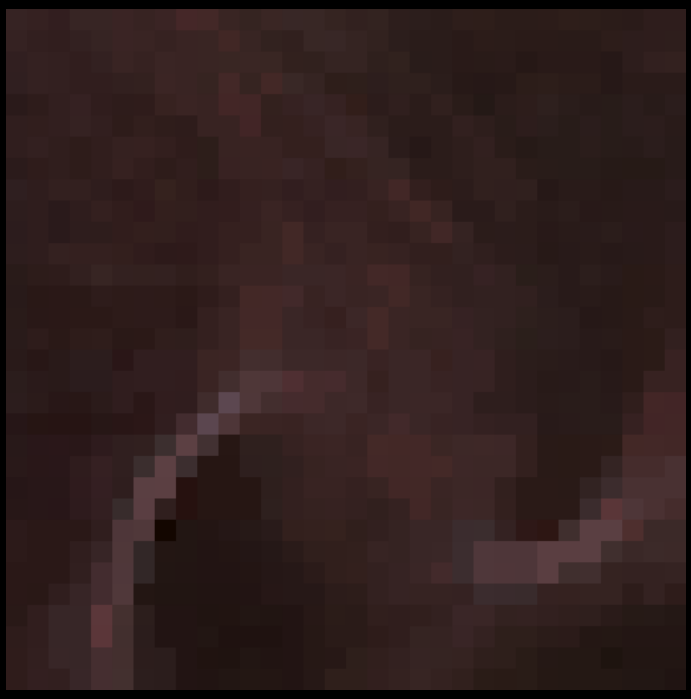
Speaker notes

This is what the target map would look like if we used it directly in game with BC1 compression. RMSE is root mean squared error, based on 8-bit unorm encoding (0-255), so lower is better.

Detail Maps | Precision

GHOST

Standard Overlay Blend RMSE: 1.74



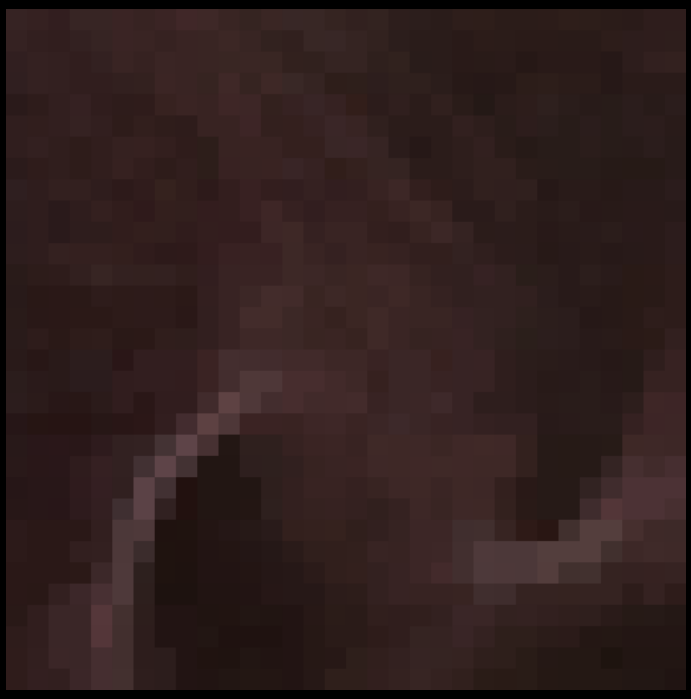
Bonus Slide

Speaker notes

Using standard overlay blend results in some ugly color banding, both in the zoomed area,
[next]
as well as in the circled area here. (Note that the error for the entire map is actually lower than the BC1 compressed target map, however! The error is higher for the parts of the image mentioned, but lower overall.)

Detail Maps | Precision

Smooth Overlay Blend (Ours) RMSE: 1.60



Bonus Slide

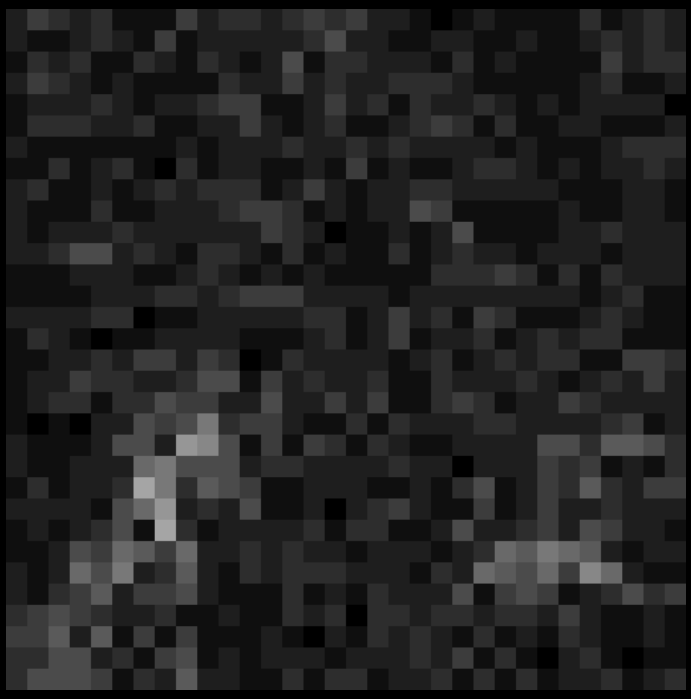
Speaker notes

Using our smooth overlay blend operator eliminates the banding and color shifts, and results in a lower error value.

Detail Maps | Precision

Target BC1 Error

RMSE: 1.95



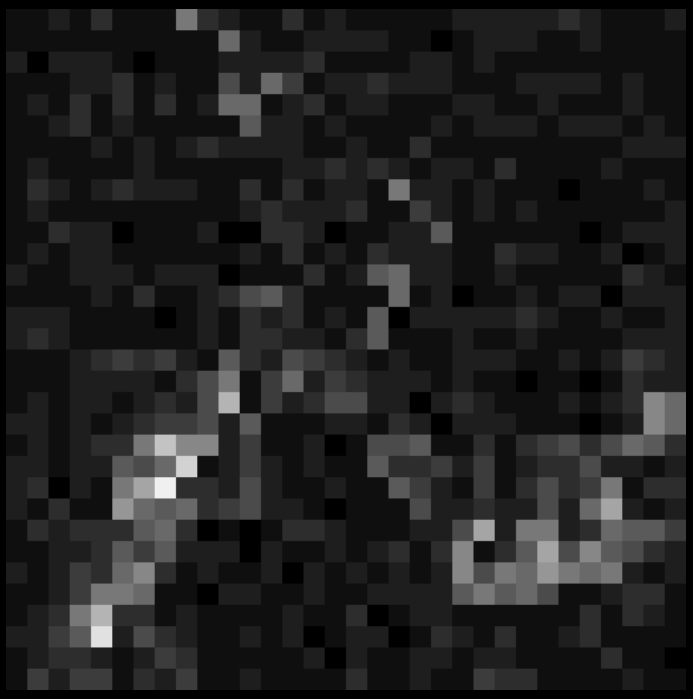
Bonus Slide

Speaker notes

Looking at just the maximum RGB error, BC1 compression of the target map looks like this.

Detail Maps | Precision

Standard Overlay Blend Error RMSE: 1.74



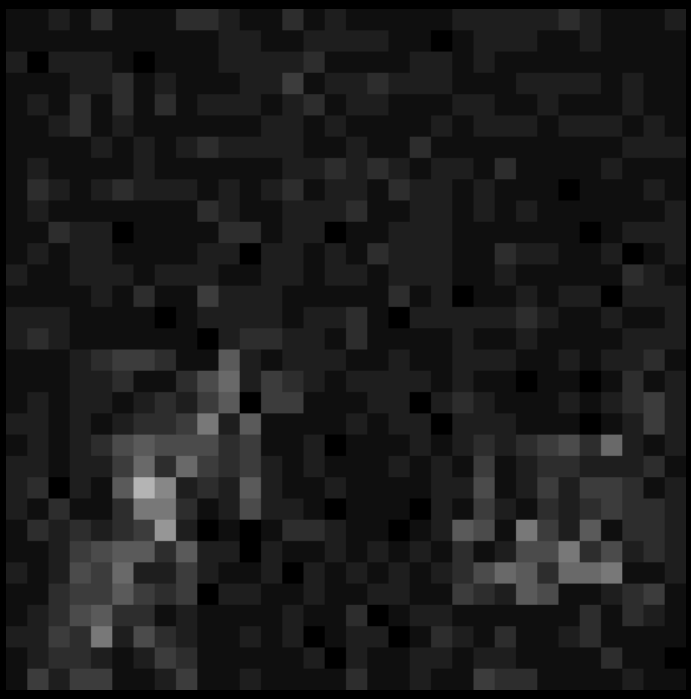
Bonus Slide

Speaker notes

Standard overlay blending looks error looks like this; note that the error is lower in many parts of the image, but also higher in other parts.

Detail Maps | Precision

Smooth Overlay Blend Error (Ours) RMSE: 1.60



Bonus Slide

Speaker notes

Using the smooth overlay blend operator reduces the error noticeably.

Detail Maps | Compression

- Blending two sets of compressed textures
  - Exacerbates compression artifacts
- Reduce by synthesizing texture against *compressed* tiled maps
  - Except for top mip which is magnified against multiple detail mips

Bonus Slide

Speaker notes

As I mentioned earlier, blending two sets of compressed textures can exacerbate compression artifacts, because the compression errors are uncorrelated.
[next]
By generating the synthesized texture against the compressed tiled maps, we can make the individual compression errors more negatively correlated, which reduces the total error after blending.
[next]
We can do this for all mips except the top (largest) mip, since that mip is magnified against multiple detail mips.

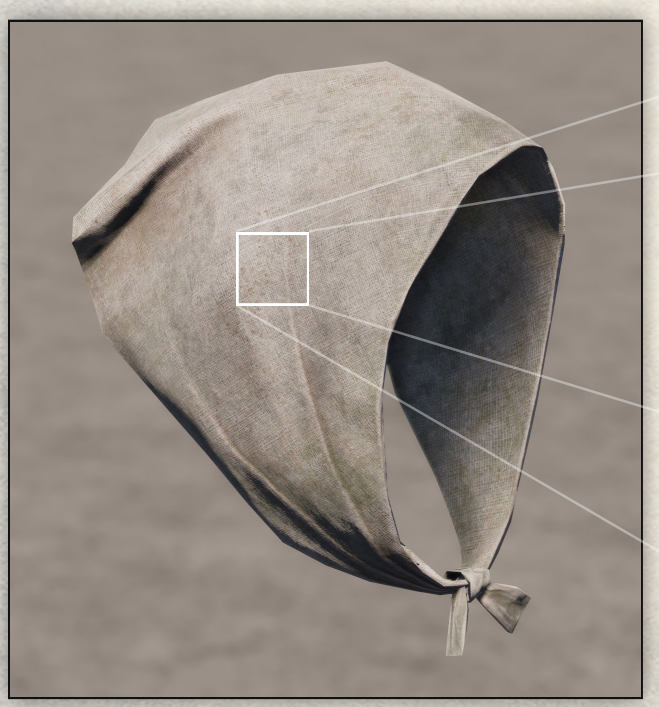
Detail Maps | Compression

Bonus Slide

Speaker notes

To demonstrate this, we'll use a different example from a scarf asset.

Detail Maps | Compression

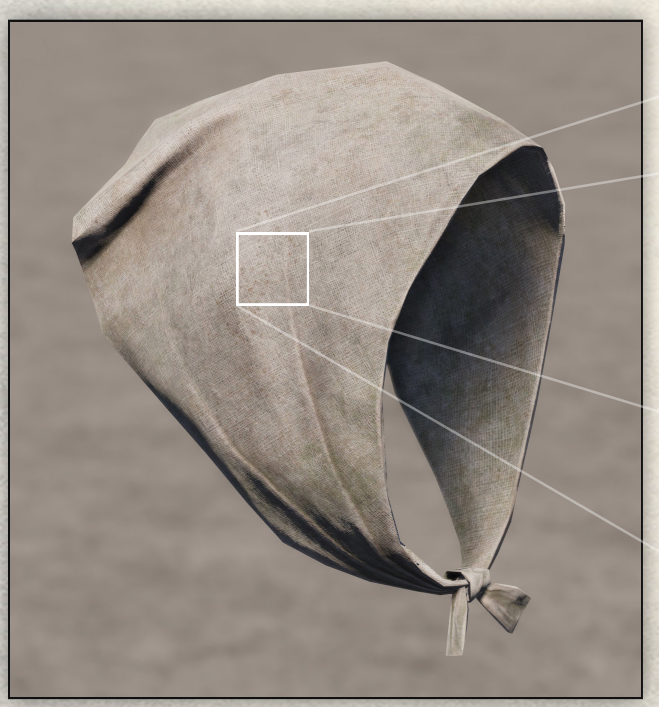


Bonus Slide

Speaker notes

This is a zoom of the blended synthesized and detail layers.

Detail Maps | Compression

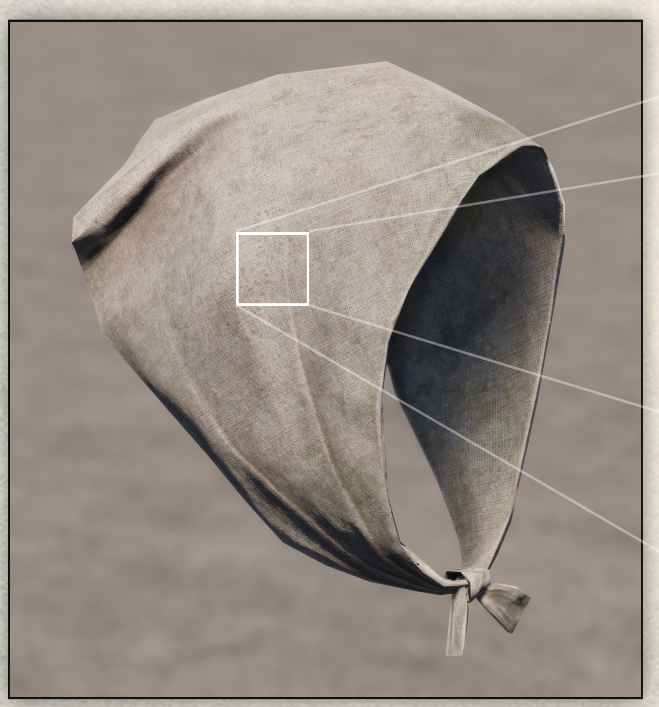


Bonus Slide

Speaker notes

This is the result without a detail layer...

Detail Maps | Compression

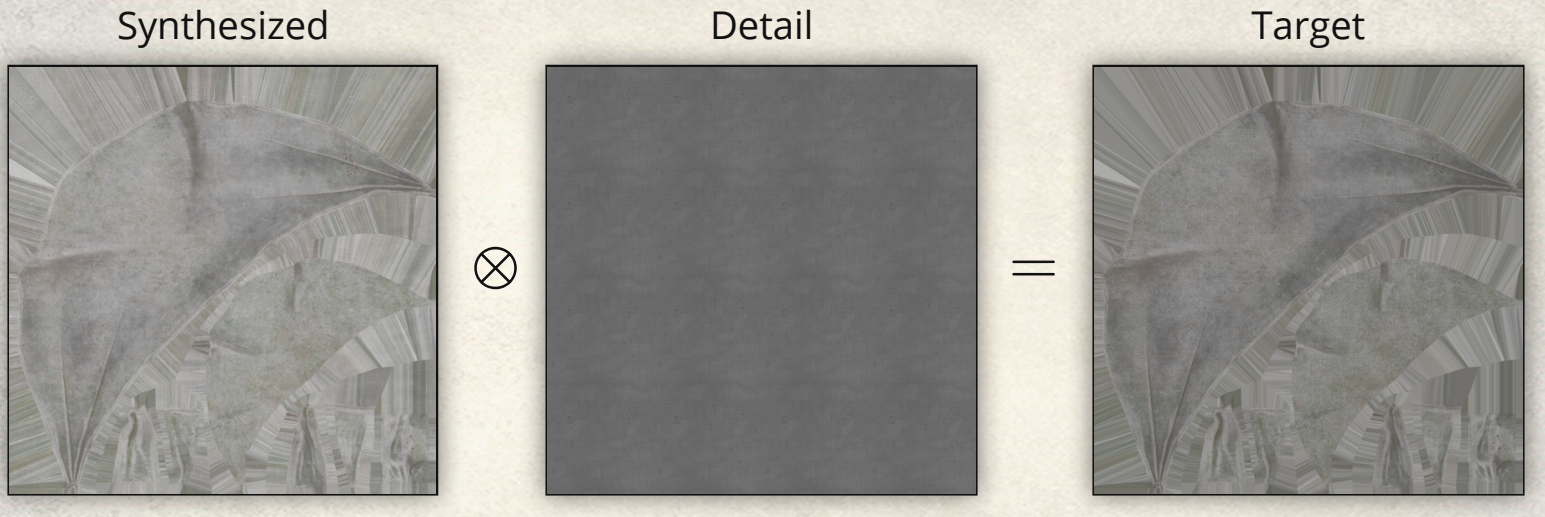


Bonus Slide

Speaker notes

... and this is the result with detail layer only.

Detail Maps | Compression



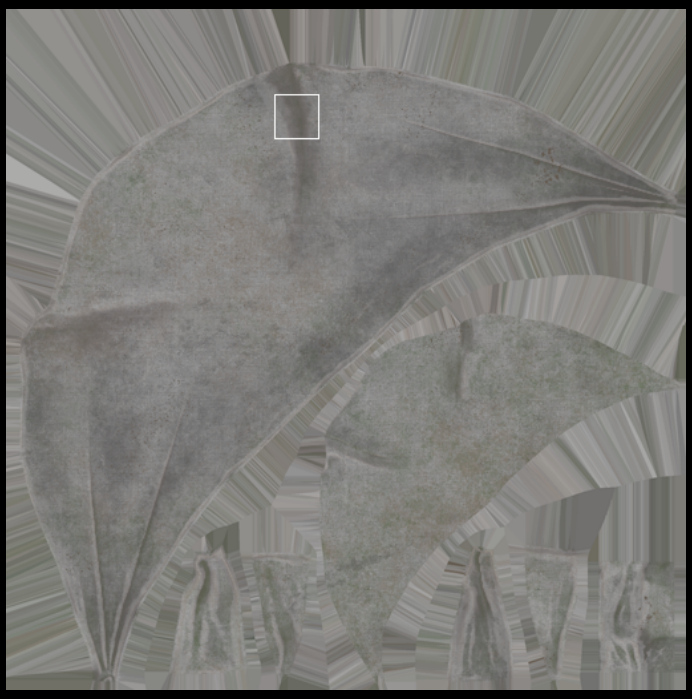
Bonus Slide

Speaker notes

These are the synthesized, detail, and target albedo maps for the scarf material; again only the synthesized and detail maps are used at runtime.

Detail Maps | Compression

Target



Bonus Slide

Speaker notes

This is the uncompressed target map, authored by the artists.
[next]
The zoomed area is shown by the white box in the left image.

Detail Maps | Compression

Blend without Compression Compensation RMSE: 2.43

Bonus Slide

Speaker notes

This is the result of blending without using compression compensation; that is, by generating the synthesized map against the uncompressed detail maps.

Detail Maps | Compression

Blend with Compression Compensation (Ours) RMSE: 2.00

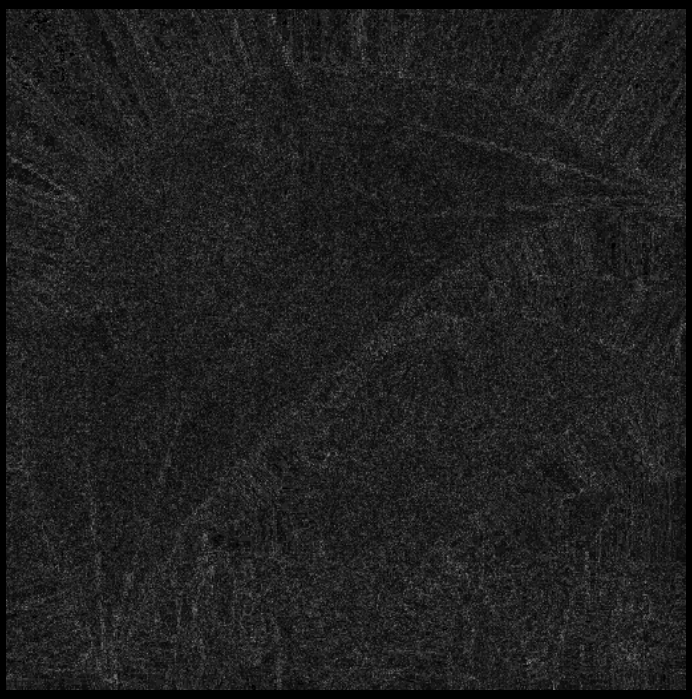
Bonus Slide

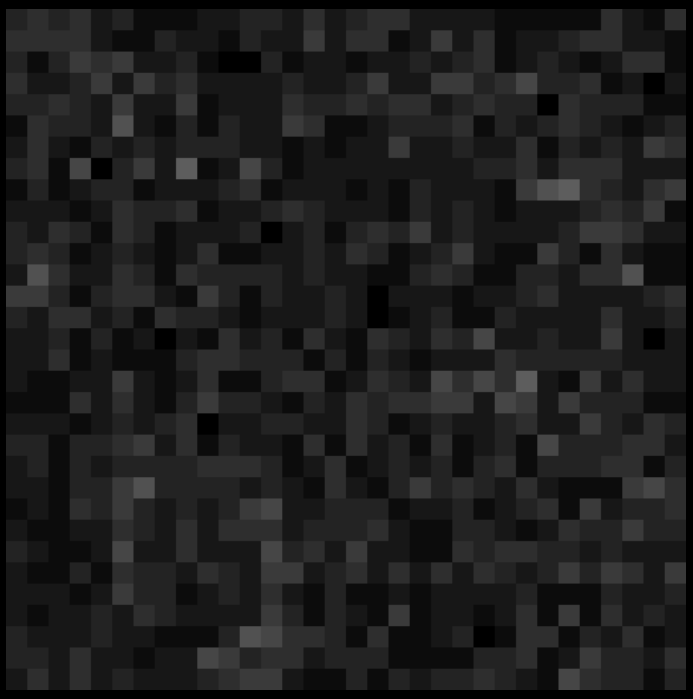
Speaker notes

This is the result with compression compensation; that is, by generating the synthesized map against the compressed detail maps. The RMS error is reduced by about 18%.

Detail Maps | Compression

Error without Compression Compensation RMSE: 2.43





Bonus Slide

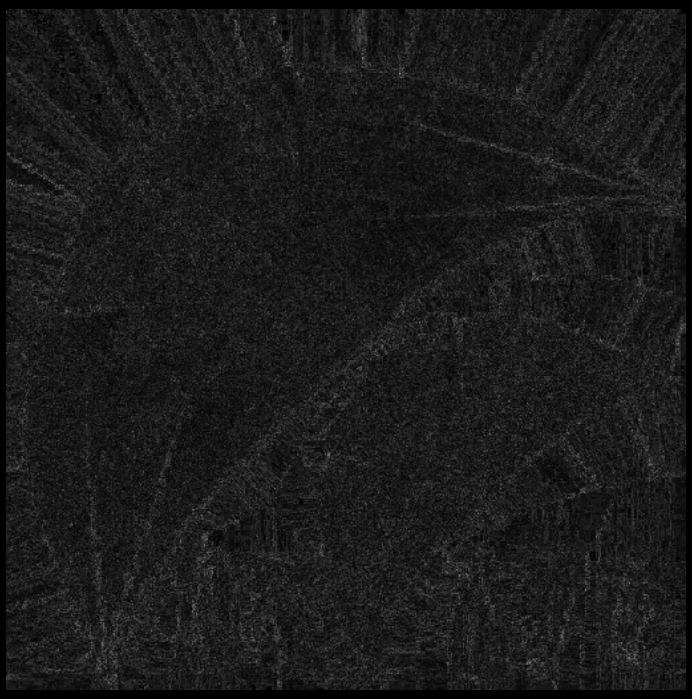
Speaker notes

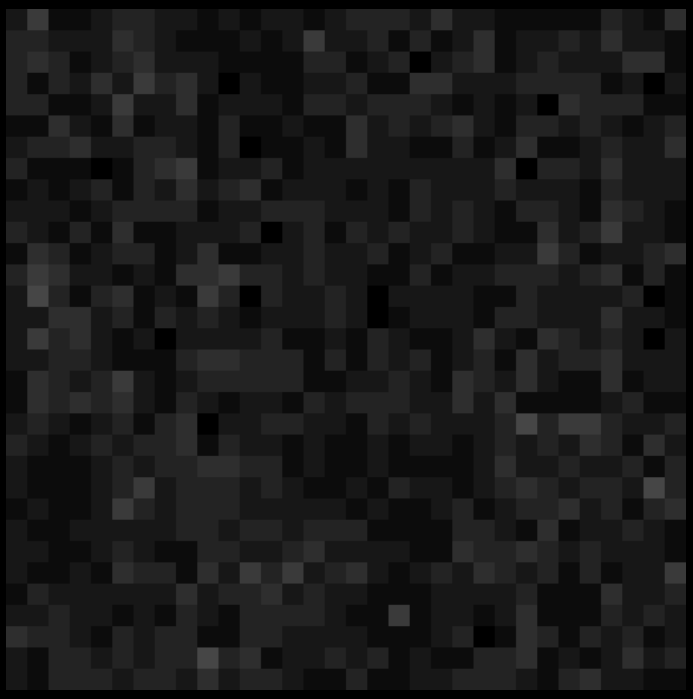
Here is the maximum RGB error without compression compensation...

Detail Maps | Compression

GHOST

Error with Compression Compensation (Ours) RMSE: 2.00





Bonus Slide

Speaker notes

... while this is the same error with compression compensation. Using compression compensation primarily helps to reduce noise in the blended result.

Detail Maps | Statistics

- Analyzed second mipmap of 1743 material color textures in game
- Average compressed target RMSE: **5.67**
- Average detail-blended RMSE: **5.81**
  - **0.14 (2.5%) higher** than with no detail
- Average RMSE using standard overlay blend: **5.89**
  - **0.09 (1.5%) higher** than ours (ΔRMSE 61% higher)
- Average RMSE without compression compensation: **5.87**
  - **0.06 (1.1%) higher** than ours (ΔRMSE 44% higher)
  - Recall that first mipmap doesn't use compression compensation

Bonus Slide

Speaker notes

Here are some statistics compiled using all of the textures using this technique in the shipping game, looking at the second mipmap level of the synthesized map only, so that we can analyze both the effect of the smooth overlay operator and compression compensation.
[next]
If we just used the compressed target map without any blending, we would get an average RMS error of 5.67.
[next]
Using our detail map approach results in an average RMS error of 5.81, which is 2.5% worse than without using detail maps. Note that we can't easily compare this to traditional detail maps, since we would have to author new detail maps. However, we expect that traditional detail maps would also have a higher RMS error compared to blending uncompressed maps and compressing the result.
[next]
Using standard overlay blending instead of smooth overlay blending results in RMS error that is 1.5% higher. Compared to the error increase from our detail blending, the error increase using standard detail blending is 61% higher.
[next]
Without compression compensation, the average RMS error is 1.1% higher, with the RMSE increase being 44% higher. Recall that compression compensation is applied to all mips except the top (largest) mip.

Detail Maps | Other Blend Modes

- Normal maps blended using Reoriented Normal Mapping (Barré-Brisebois and Hill, 2012)
  - https://blog.selfshadow.com/publications/blending-in-detail/
  - Compute like other maps (by inverting the blend function)
- Anisotropic specular (“aniso”) maps blended with regular alpha blending
  - I.e., “over” operator
  - Alpha is a shader parameter

Bonus Slide

Speaker notes

We used the smooth overlay blend operator for most types of maps, except for some special cases. Normal maps were blended using Reoriented Normal Mapping, which is a principled and practical way to applying detail normal maps.
[next]
The synthesized map was still computed by inverting the blend operator, which we did numerically using Newton's method in 2D.
[next]
Anisotropic specular maps can't be blended using overlay blend, since it doesn't preserve the structure of the matrix. We opted to use alpha blending, with the alpha parameter exposed as an adjustable shader parameter. The synthesized aniso map was again generated by inverting this blending operator, subject to the constraint that the eigenvalues of the anisotropic matrix were in the range (0, 1].

Detail Maps | Summary

- Successful widespread adoption of detail maps for Ghost
- Advantages over traditional approach:
  - Can selectively cancel out lower frequency components of detail
  - More potential texture reuse since detail can be used as material
- Disadvantages:
  - Slightly longer texture processing
  - Slightly higher RMSE on average for color textures (vs no detail)
    - Can be reduced by centering histogram of detail maps
      - I.e., same type of preprocessing used with traditional detail maps

Bonus Slide

Speaker notes

In summary, this approach to authoring detail maps proved to be very successful for Ghost, as demonstrated by the large number of textures that were authored with detail maps this way.
[next]
One advantage over the traditional detail map approach is the ability to selectively cancel out the lower frequency components of the detail textures;
[next]
another is the potential for greater texture reuse since detail maps can be used directly as materials, as they don't require preprocessing of any kind.
[next]
Generating the synthesized maps results in slightly longer texture processing times (though this did not prove to be prohibitive),
[next]
and the RMS error for color textures is slightly higher on average vs no detail.
[next]
Note that this error can be reduced by centering the histogram of detail maps, which is the same type of preprocessing usually done with traditional detail maps.

Acknowledgements

- My wife and son for their patience while I went from finishing Ghost to working on this presentation
- Sucker Punch rendering programmers:
  - Adrian Bentley
  - Bill Rockenbeck
  - Eric Wohllaib (who implemented deferred fuzziness)
  - Matthew Pohlmann
  - Tom Low
- Drew Harrison, Harold Lamb, Joanna Wang, Omar Aweidah, and Phillip Jenne for their assistance with assets for this talk
- Everybody at Sucker Punch for making everything awesome
- Steve McAuley and Stephen Hill for the insightful feedback

Speaker notes

I'd like to thank the following people for their contributions;
First I'd like to thank my family for their patience while I went from working hard finishing Ghost, to then working hard on this talk;
I'd also like to thank the Sucker Punch rendering team for all of their hard work;
I'd like to thank members of the art team for their help with assets for this talk;
as well as everybody at Sucker Punch for making it an awesome place to work;
and finally I'd like to thank Steve and Steve for their valuable feedback.

References

[DHI+13] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. Linear efficient antialiased displacement and reflectance mapping. *ACM Trans. Graph.*, 32(6), November 2013.

[HDCD15] Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. The SGGX microflake distribution. *ACM Trans. Graph.*, 34(4), July 2015.

[KK89] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *SIGGRAPH Comput. Graph.*, 23(3):271–280, July 1989.

[KP03] Jan Koenderink and Sylvia Pont. The secret of velvety skin. *Machine Vision and Applications*, 14(4):260–268, 2003.

[McA13] Steve McAuley. Extension to energy-conserving wrapped diffuse. http://blog.stevemcauley.com/2013/01/30/extension-to-energy-conserving-wrapped-diffuse/, January 2013.

[MHH+12] Stephen McAuley, Stephen Hill, Naty Hoffman, Yoshiharu Gotanda, Brian Smits, Brent Burley, and Adam Martinez. Practical physically-based shading in film and game production. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, New York, NY, USA, 2012. Association for Computing Machinery.

[OB10] Marc Olano and Dan Baker. Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, page 181–188, New York, NY, USA, 2010. Association for Computing Machinery.

[PB11] Eric Penner and George Borshukov. Pre-integrated skin shading. In Wolfgang Engel, editor, *GPU Pro 2*, chapter 1. A K Peters/CRC Press, New York, 1st edition, 2011.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*, pages 583–587. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[RH01a] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 497–500, New York, NY, USA, 2001. ACM.

[RH01b] Ravi Ramamoorthi and Pat Hanrahan. On the relationship between Radiance and Irradiance: Determining the illumination from images of a convex Lambertian object. *Journal of the Optical Society of America*, 18(10):2448–2459, Oct 2001.

[Rus04] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*, September 2004.

[RV11] P. Ramachandran and G. Varoquaux. Mayavi: 3D Visualization of Scientific Data. *Computing in Science & Engineering*, 13(2):40–51, 2011.

Speaker notes

Here are the references for this talk; which you'll be able to consult online;

Speaker notes

And finally, I'd like to thank *you* very much for your attention.