SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

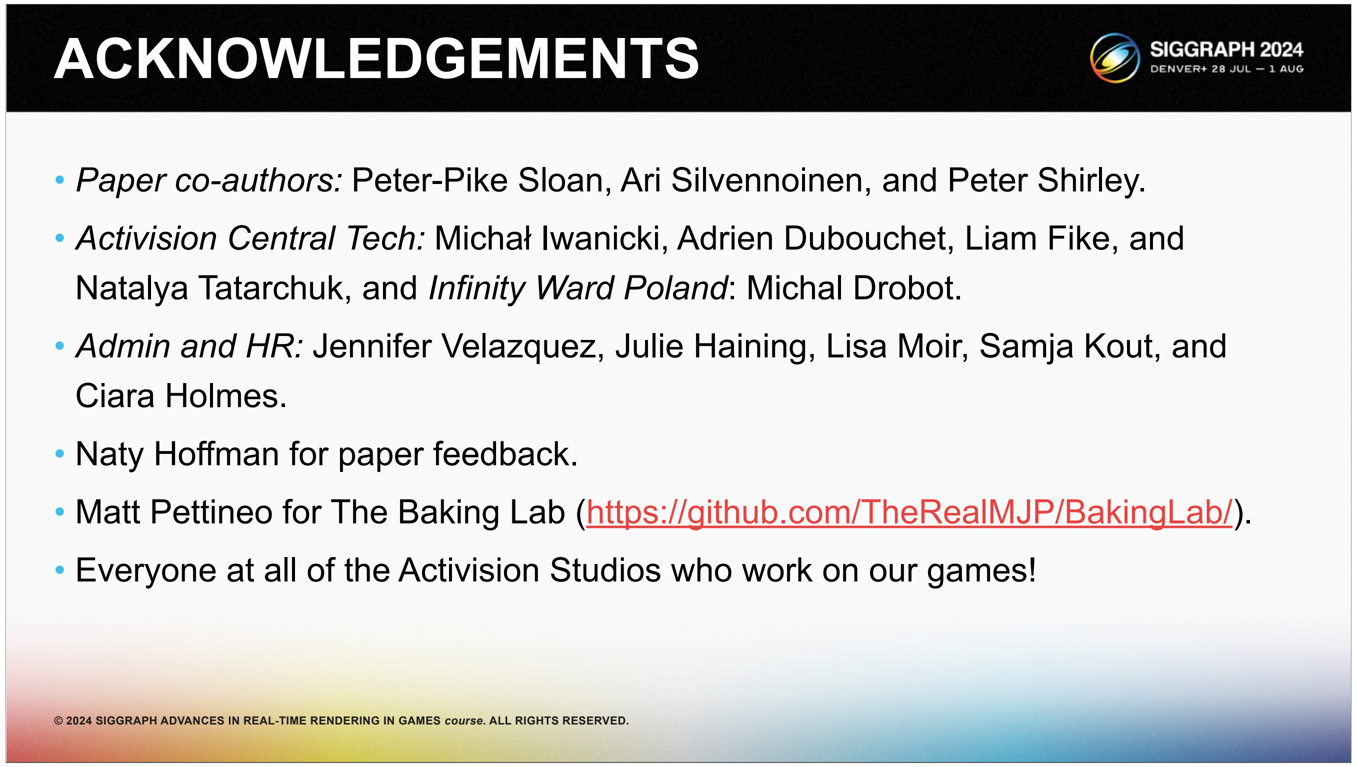
THE PREMIER CONFERENCE
& EXHIBITION ON
COMPUTER GRAPHICS &
INTERACTIVE TECHNIQUES

HEMISPHERICAL LIGHTING INSIGHTS

FROM THE *CALL OF DUTY* PRODUCTION LESSONS

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Hi. I’m Thomas. I’m a graphics engineer at Activision Central Tech working on lighting for Call of Duty, and today I’m going to be discussing some recent developments and insights from our secondary lighting pipeline.



ACKNOWLEDGEMENTS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

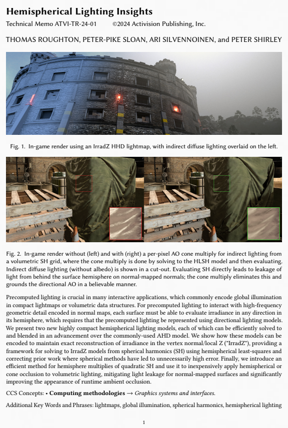
- *Paper co-authors:* Peter-Pike Sloan, Ari Silvennoinen, and Peter Shirley.
- *Activision Central Tech:* Michał Iwanicki, Adrien Dubouchet, Liam Fike, and Natalya Tatarchuk, and *Infinity Ward Poland*: Michal Drobot.
- *Admin and HR:* Jennifer Velazquez, Julie Haining, Lisa Moir, Samja Kout, and Ciara Holmes.
- Naty Hoffman for paper feedback.
- Matt Pettineo for The Baking Lab (https://github.com/TheRealMJP/BakingLab/).
- Everyone at all of the Activision Studios who work on our games!

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

This talk today is built on many years of work from many people. I want to particularly thank the co-authors for the paper this is based on: Peter-Pike Sloan, Ari Silvennoinen, and Peter Shirley.

TOPICS

- Optimal lightmap solves
- Hemispherical lighting model (HHD) for lightmap blending
- Fast hemispherical occlusion for SH
- Fast cone occlusion from runtime AO for SH



Hemispherical Lighting Insights

Technical Memo ATVI-TR-24-01 ©2024 Activision Publishing, Inc.

THOMAS ROUGHTON, PETER-PIKE SLOAN, ARI SILVENNOINEN, and PETER SHIRLEY

Fig. 1. In-game render using an IrradZ HHD lightmap, with indirect diffuse lighting overlaid on the left.

Fig. 2. In-game render without (left) and with (right) a per-pixel AO cone multiply for indirect lighting from a volumetric SH grid, where the cone multiply is done by solving to the HLSH model and then evaluating. Indirect diffuse lighting (without albedo) is shown in a cut-out. Evaluating SH directly leads to leakage of light from behind the surface hemisphere on normal-mapped normals; the cone multiply eliminates this and grounds the directional AO in a believable manner.

Precomputed lighting is crucial in many interactive applications, which commonly encode global illumination in compact lightmaps or volumetric data structures. For precomputed lighting to interact with high-frequency geometric detail encoded in normal maps, each surface must be able to evaluate irradiance in any direction in its hemisphere, which requires that the precomputed lighting be represented using directional lighting models. We present two new highly compact hemispherical lighting models, each of which can be efficiently solved to and blended in an advancement over the commonly-used AHD model. We show how these models can be encoded to maintain exact reconstruction of irradiance in the vertex normal/local Z ("IrradZ"), providing a framework for solving to IrradZ models from spherical harmonics (SH) using hemispherical least-squares and correcting prior work where spherical methods have led to unnecessarily high error. Finally, we introduce an efficient method for hemisphere multiplies of quadratic SH and use it to inexpensively apply hemispherical or cone occlusion to volumetric lighting, mitigating light leakage for normal-mapped surfaces and significantly improving the appearance of runtime ambient occlusion.

CCS Concepts: • **Computing methodologies** → *Graphics systems and interfaces.*

Additional Key Words and Phrases: lightmaps, global illumination, spherical harmonics, hemispherical lighting

1

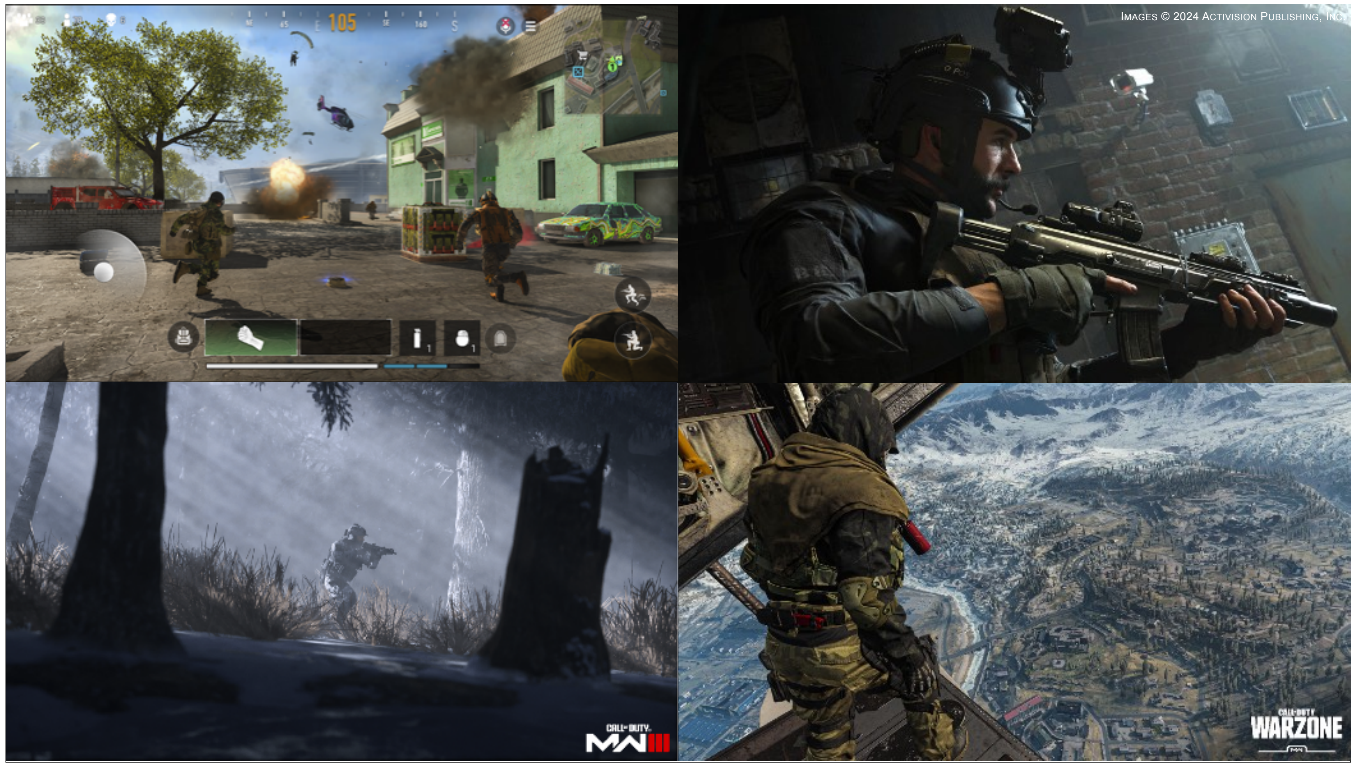
https://research.activision.com/

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

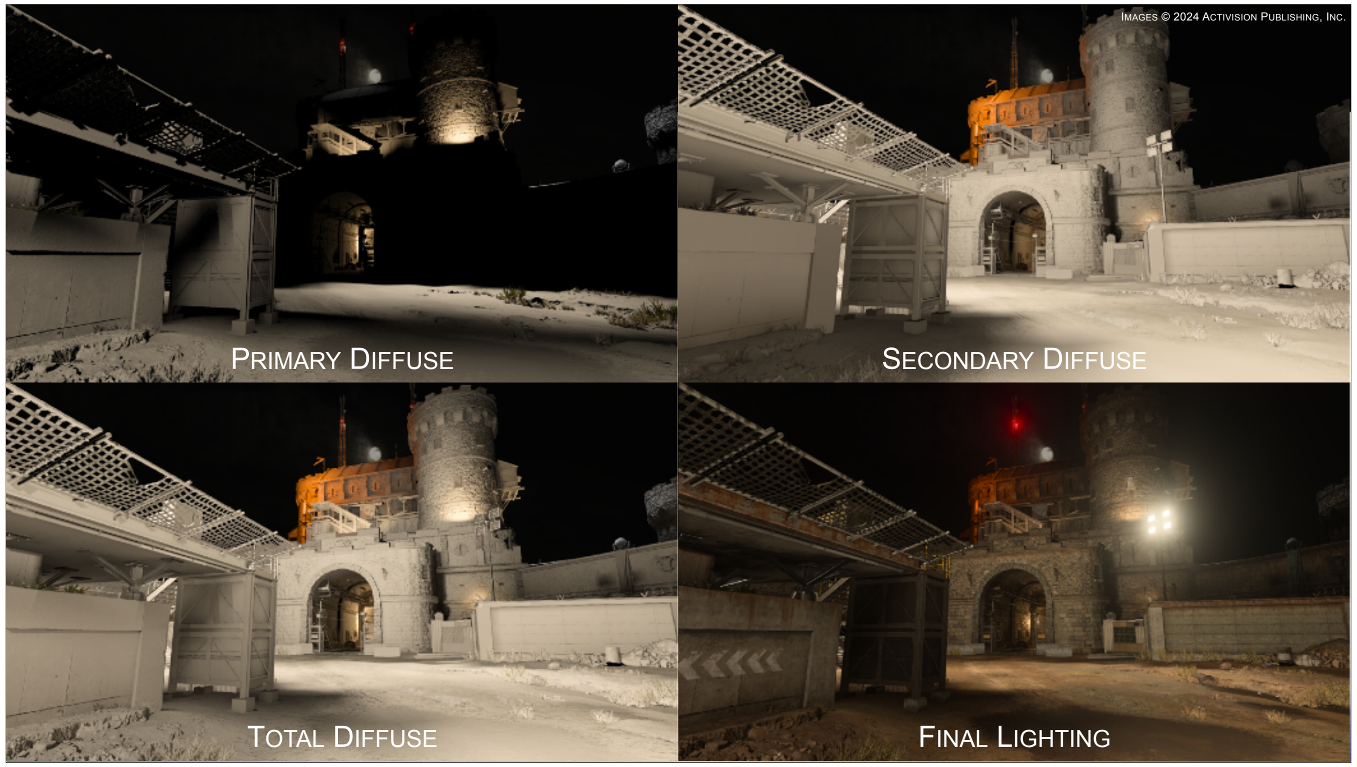
The main topics I'll cover today are:

- How we got better quality lightmaps using our existing runtime format through improving how we encode them.
- Our new hemispherical lighting model that allows for more accurate lightmap blending.
- New, fast techniques to compute hemispherical occlusion for volumetric spherical lighting, and
- A new, efficient way to compute occlusion by a visibility cone for runtime AO.

These techniques were published in an Activision technical memo in May, so if you're curious for more details after the talk I'd suggest giving that a look.

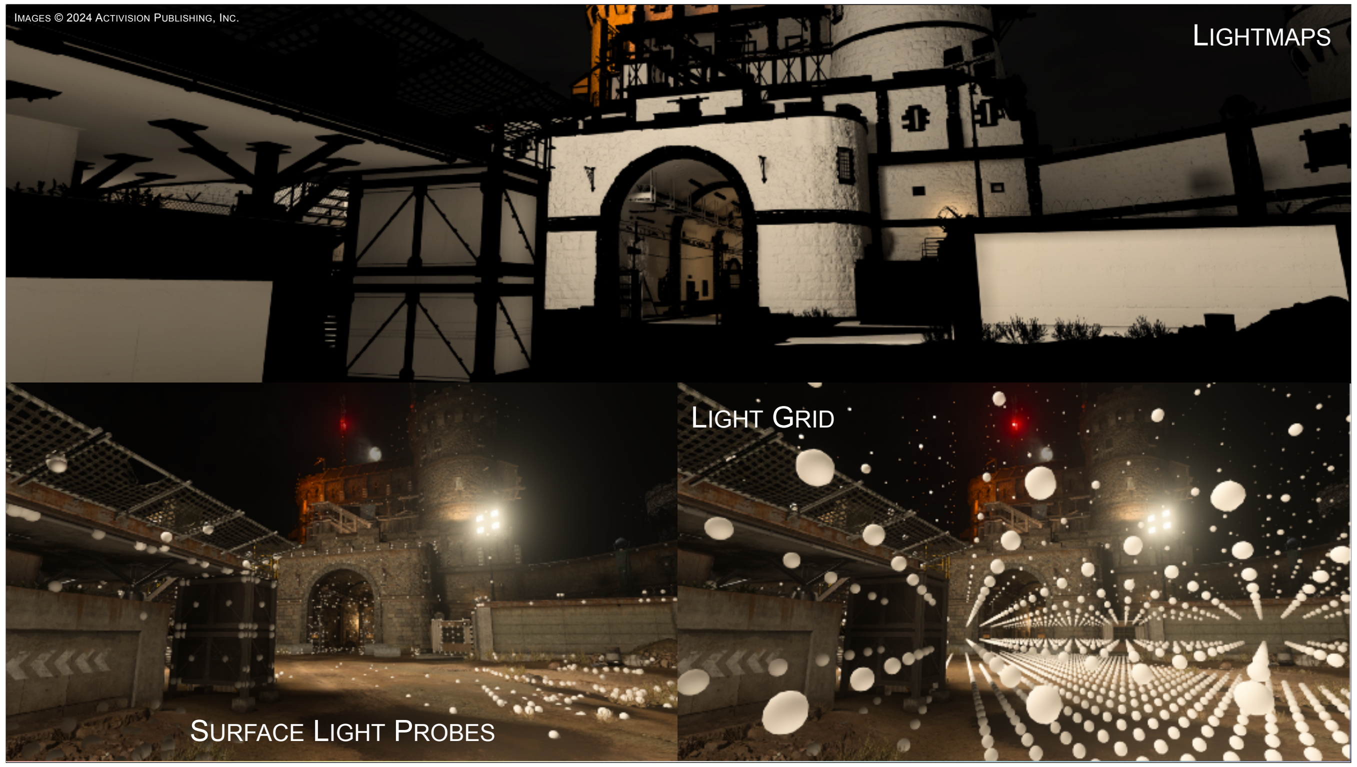

Images © 2024 Activision Publishing, Inc.

Before we get into the details, let’s set out some context. Call of Duty is a fast-paced first person shooter that runs on a wide range of hardware, from low-end mobile phones to current generation consoles to high end PCs running at hundreds of frames per second. We support a wide range of content, from dense indoor environments to the vast scale of Warzone. This means that scalability is hugely important to us. When we’re implementing lighting techniques, we’re thinking about how they can scale across that performance range to deliver a great experience on all platforms, while still providing cutting-edge visuals on the latest hardware.



For that reason, our lighting pipeline is heavily reliant on baked lighting for both direct and indirect illumination; everything you see under secondary diffuse here is coming from baked lighting. Much of that is indirect bounce, but it also includes the sky, emissive materials, and static and residual lights.

The diffuse images are combined with the albedo, specular lighting, and post-processing to produce the final result.



Images © 2024 Activision Publishing, Inc.

Our baked lighting is split between three different sources. Static, simple objects use lightmaps; objects that have a more difficult lightmap parameterisation use a volume of local light probes, while dynamic objects resample a local volume from a grid of probes.

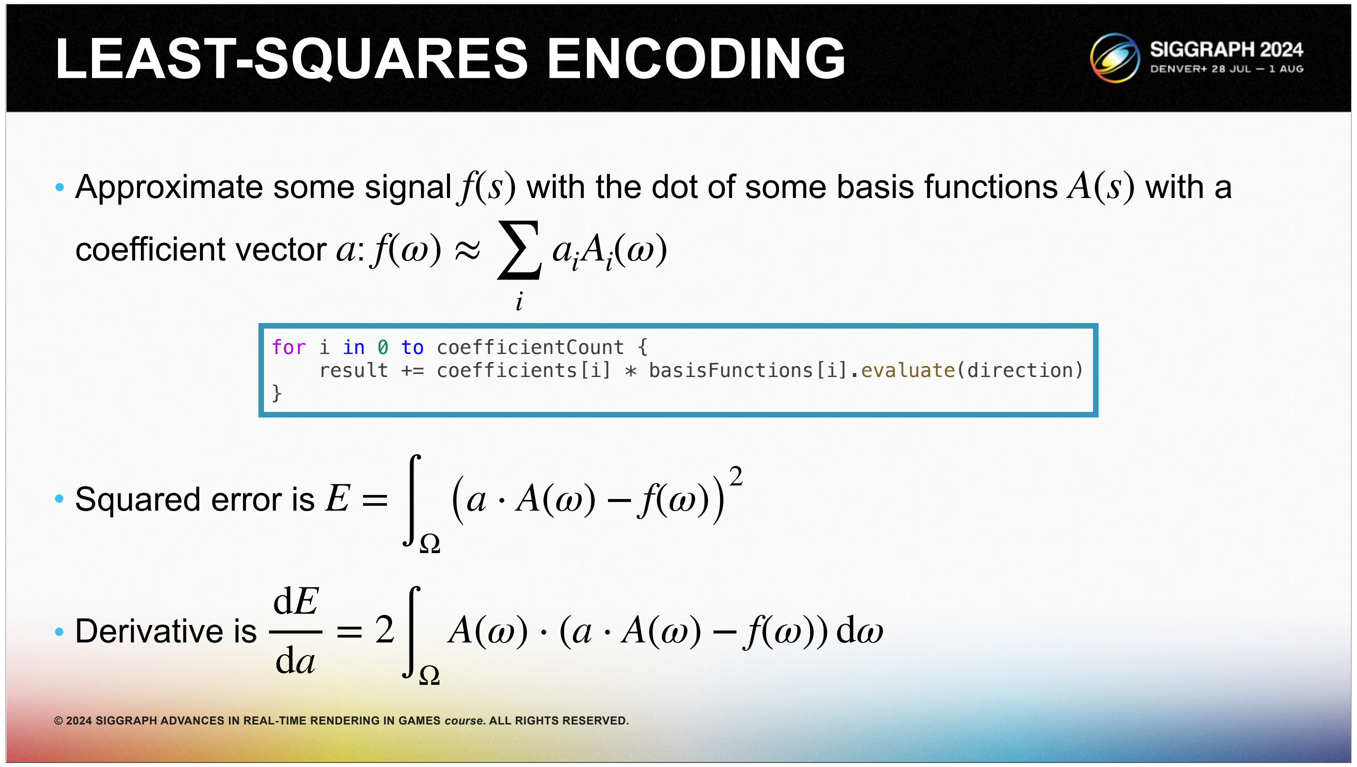
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

SPHERICAL HARMONICS

AND LEAST-SQUARES ENCODING

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Those probes are encoded using spherical harmonics, and our lightmap bake also uses spherical harmonics heavily. Spherical harmonics can get a bit of a scary reputation, so I want to provide a quick grounding in some of the maths behind them.



LEAST-SQUARES ENCODING

- Approximate some signal $f(s)$ with the dot of some basis functions $A(s)$ with a coefficient vector a: $f(\omega) \approx \sum_i a_i A_i(\omega)$

```
for i in 0 to coefficientCount {
    result += coefficients[i] * basisFunctions[i].evaluate(direction)
}
```

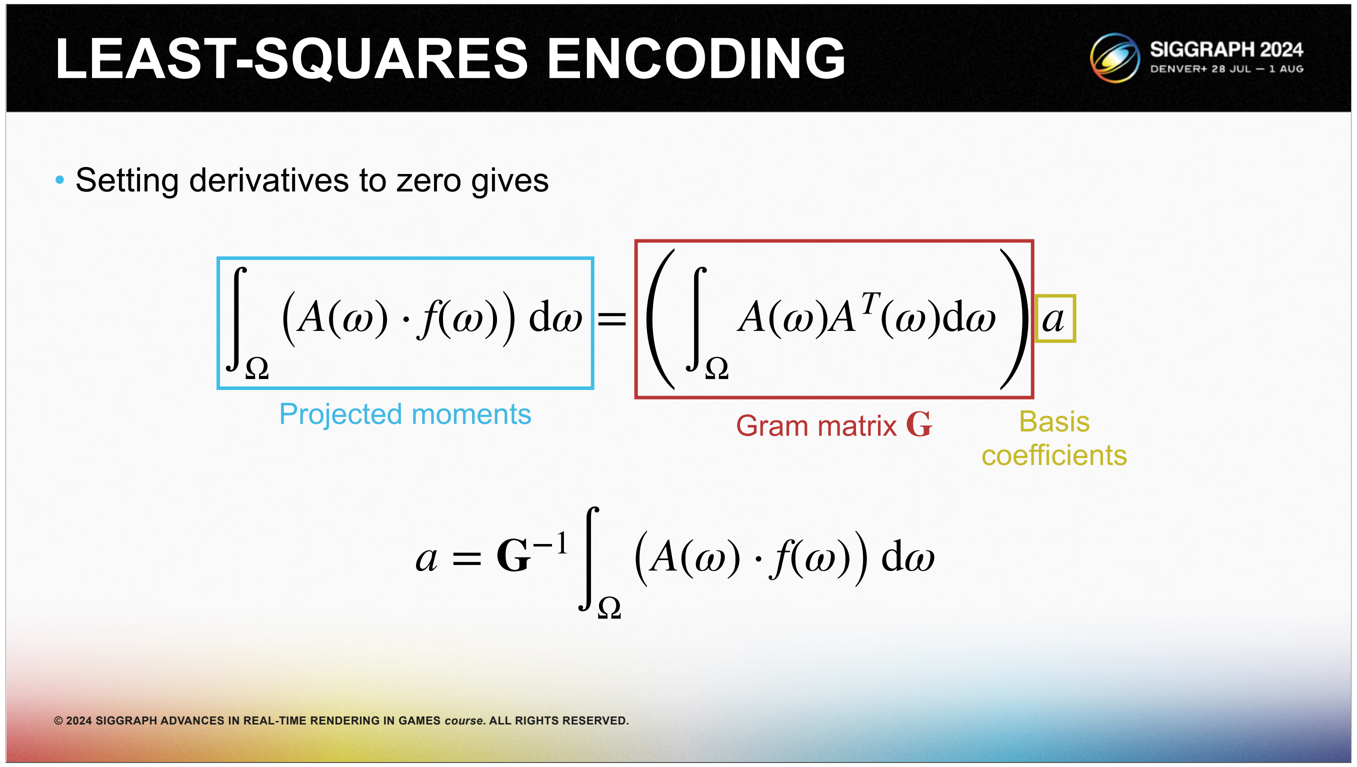
- Squared error is $E = \int_\Omega \left(a \cdot A(\omega) - f(\omega)\right)^2$
- Derivative is $\frac{\mathrm{d}E}{\mathrm{d}a} = 2\int_\Omega A(\omega) \cdot (a \cdot A(\omega) - f(\omega))\,\mathrm{d}\omega$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

I find it easiest to approach spherical harmonics by starting from the concept of a linear basis.

We want to approximate some signal – say radiance or irradiance – for all directions on a sphere or hemisphere. We do that by defining what are called basis functions – functions that will give us some value for some direction – and computing a coefficient vector that gives weights for those functions. To reconstruct the original signal, we just sum the weighted basis functions for the direction we're querying.

Computing the coefficient vector in a least-squares sense uses standard least-squares techniques. We define the error to be the squared difference between our reconstructed value and the target function, and the derivative is our basis vector times the difference.



LEAST-SQUARES ENCODING

- Setting derivatives to zero gives

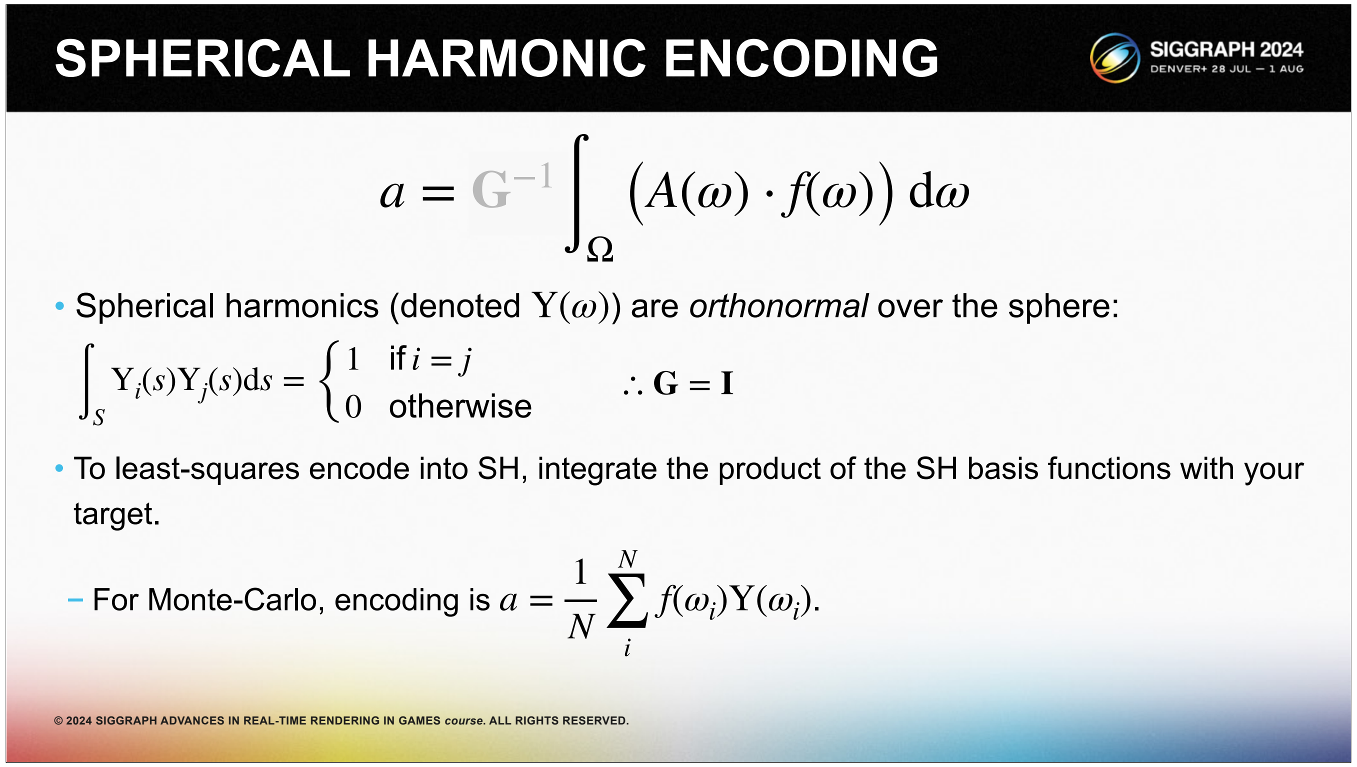
$$\underset{\text{Projected moments}}{\boxed{\int_\Omega \big(A(\omega)\cdot f(\omega)\big)\,\mathrm{d}\omega}} = \underset{\text{Gram matrix }\mathbf{G}}{\boxed{\left(\int_\Omega A(\omega)A^T(\omega)\mathrm{d}\omega\right)}}\underset{\text{Basis coefficients}}{\boxed{a}}$$

$$a = \mathbf{G}^{-1}\int_\Omega \big(A(\omega)\cdot f(\omega)\big)\,\mathrm{d}\omega$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Setting the derivatives to zero, we get this equation. On the left hand side we have what are called the moments; this is the projection of the source signal against the basis functions. On the right, we have what’s known as the Gram matrix, which roughly tells you how much the basis functions overlap over the domain, along with our coefficient vector *a*. To solve for *a*, you can multiply the basis coefficients by the inverse of the Gram matrix.

This works for encoding into any linear basis, although the Gram matrix inverse may be more or less well-behaved depending on the source functions and the integration domain.



SPHERICAL HARMONIC ENCODING

$$a = \mathbf{G}^{-1} \int_{\Omega} \left(A(\omega) \cdot f(\omega) \right) \mathrm{d}\omega$$

- Spherical harmonics (denoted $\mathrm{Y}(\omega)$) are *orthonormal* over the sphere:

$$\int_{S} \mathrm{Y}_i(s)\mathrm{Y}_j(s)\mathrm{d}s = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \therefore \mathbf{G} = \mathbf{I}$$

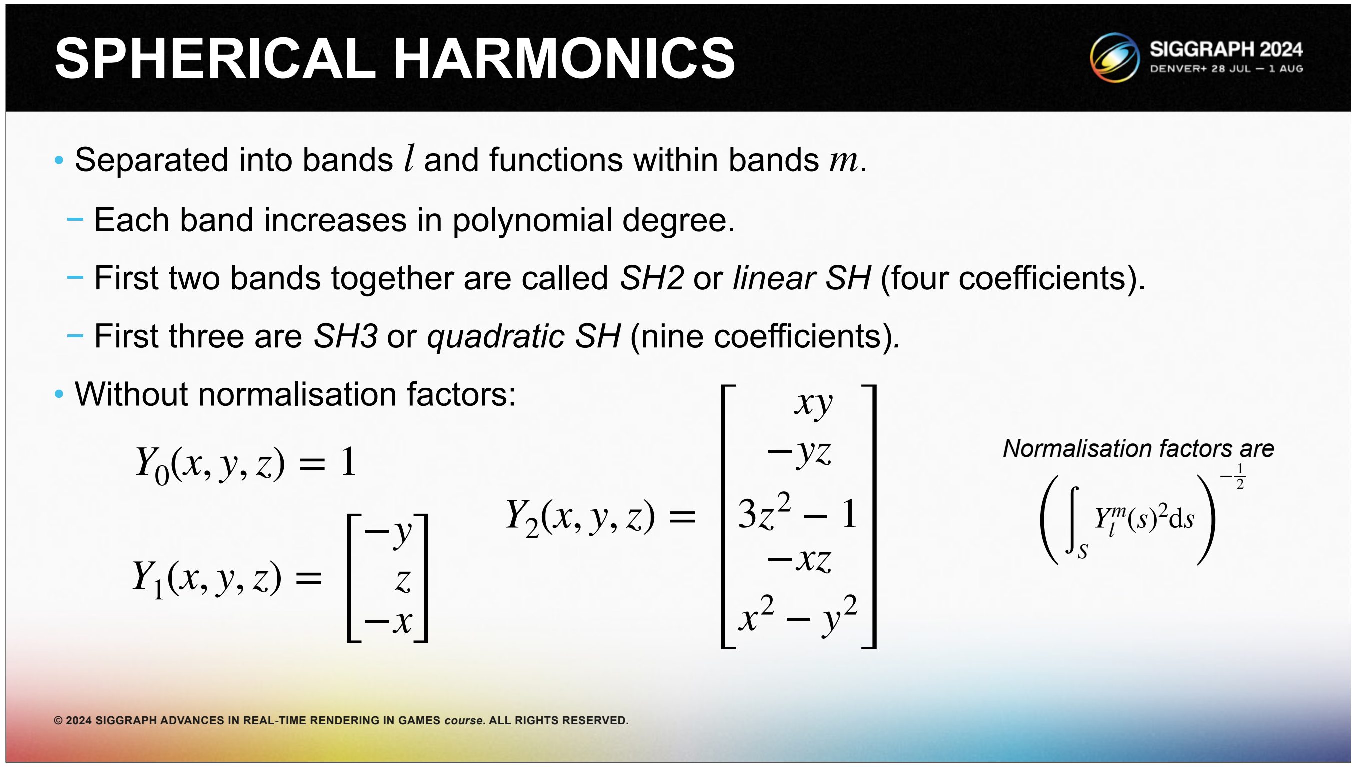
- To least-squares encode into SH, integrate the product of the SH basis functions with your target.
  - For Monte-Carlo, encoding is $a = \frac{1}{N}\sum_{i}^{N} f(\omega_i)\mathrm{Y}(\omega_i)$.

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Spherical harmonics are special in that they’re what’s known as *orthonormal* over the sphere; the spherical integral of the product of any two functions in the set will be 1 if they’re the same function or zero if they’re not. This means the spherical Gram matrix is the identity matrix, and so the least-squares coefficient vector is just the moments.

Therefore, to least-squares encode into spherical harmonics, you simply integrate the product of the SH basis functions with your target function. If you’re using Monte-Carlo, you just evaluate the basis in the sample direction, multiply by the sample intensity, and then sum the results.

Note that this only applies to minimise error over a spherical domain; this will be important later.



SPHERICAL HARMONICS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Separated into bands l and functions within bands m.
  - Each band increases in polynomial degree.
  - First two bands together are called *SH2* or *linear SH* (four coefficients).
  - First three are *SH3* or *quadratic SH* (nine coefficients).
- Without normalisation factors:

$$Y_0(x, y, z) = 1$$

$$Y_1(x, y, z) = \begin{bmatrix} -y \\ z \\ -x \end{bmatrix}$$

$$Y_2(x, y, z) = \begin{bmatrix} xy \\ -yz \\ 3z^2 - 1 \\ -xz \\ x^2 - y^2 \end{bmatrix}$$

Normalisation factors are

$$\left(\int_S Y_l^m(s)^2 \mathrm{d}s \right)^{-\frac{1}{2}}$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Spherical harmonics are separated into bands, and each band is of increasing polynomial degree. For lighting, we usually only use the first two or three bands, referred to as linear SH for two bands or quadratic SH for three.

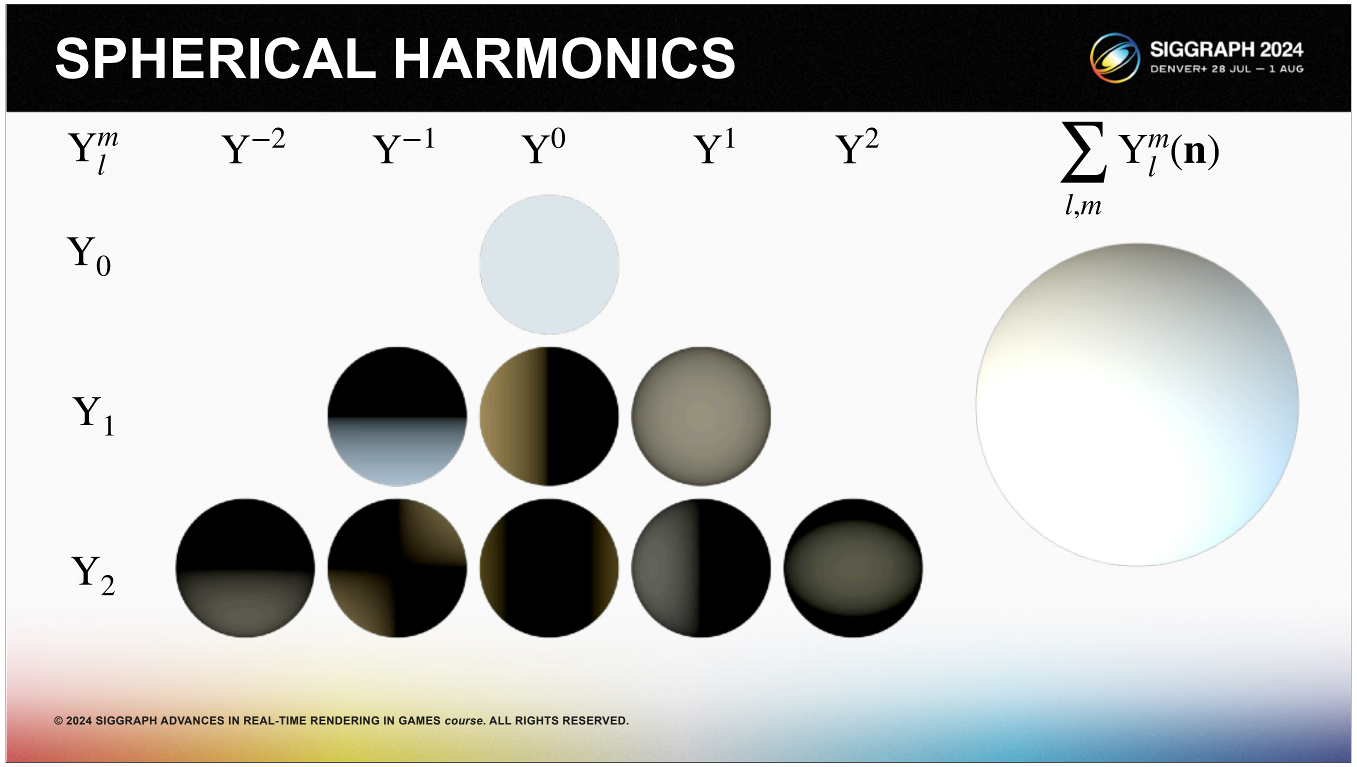
Without the normalisation factors, these functions are pretty simple.

The L0 band is just a constant.

The L1 band is a rearranged version of the input vector.

And the L2 band has combinations of each of the coefficients.

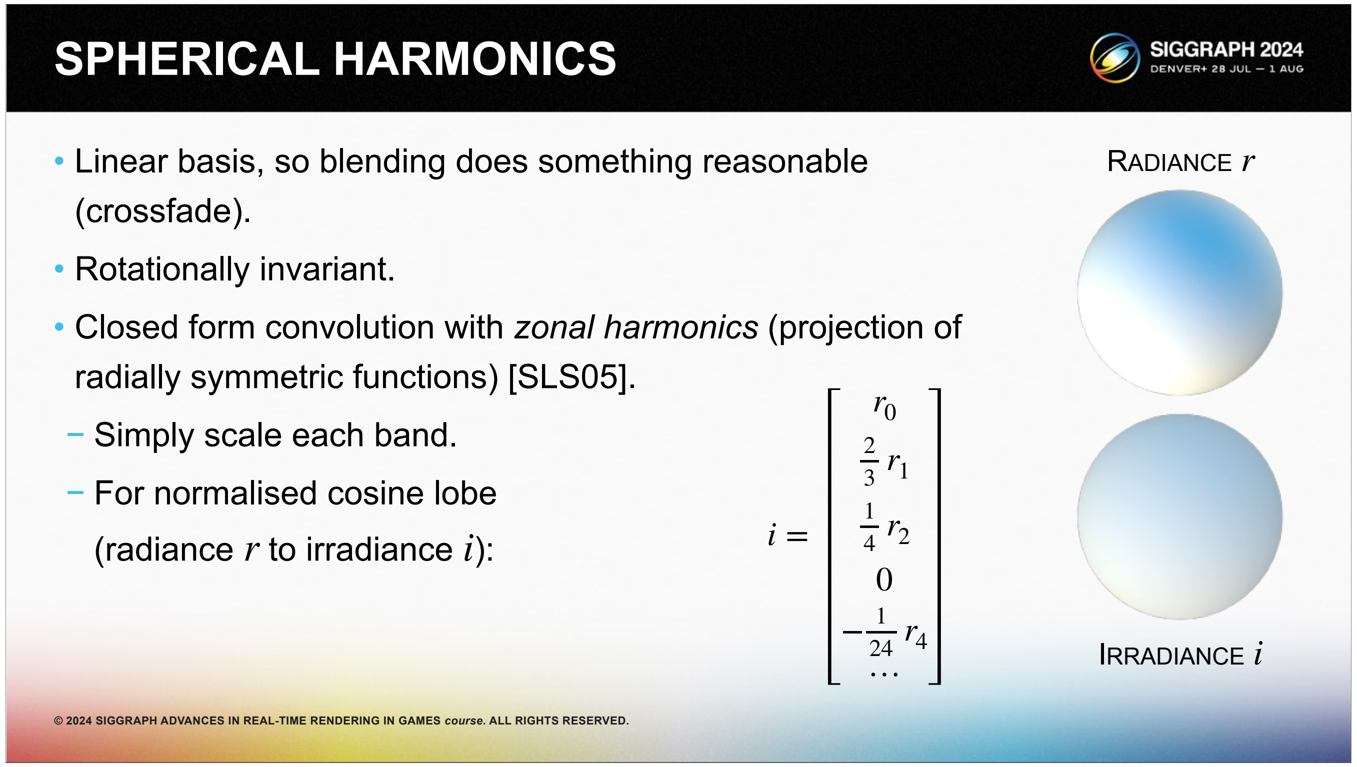
In practice, there are also scale factors for each function, which you can derive from the orthonormality property.



SPHERICAL HARMONICS

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

This shows the reconstruction of a quadratic spherical harmonics radiance probe from its individual basis functions and coefficients. These functions can go negative, which is represented here with black.



SPHERICAL HARMONICS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Linear basis, so blending does something reasonable (crossfade).
- Rotationally invariant.
- Closed form convolution with *zonal harmonics* (projection of radially symmetric functions) [SLS05].
  - Simply scale each band.
  - For normalised cosine lobe (radiance r to irradiance i):

$$i = \begin{bmatrix} r_0 \\ \frac{2}{3} r_1 \\ \frac{1}{4} r_2 \\ 0 \\ -\frac{1}{24} r_4 \\ \cdots \end{bmatrix}$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

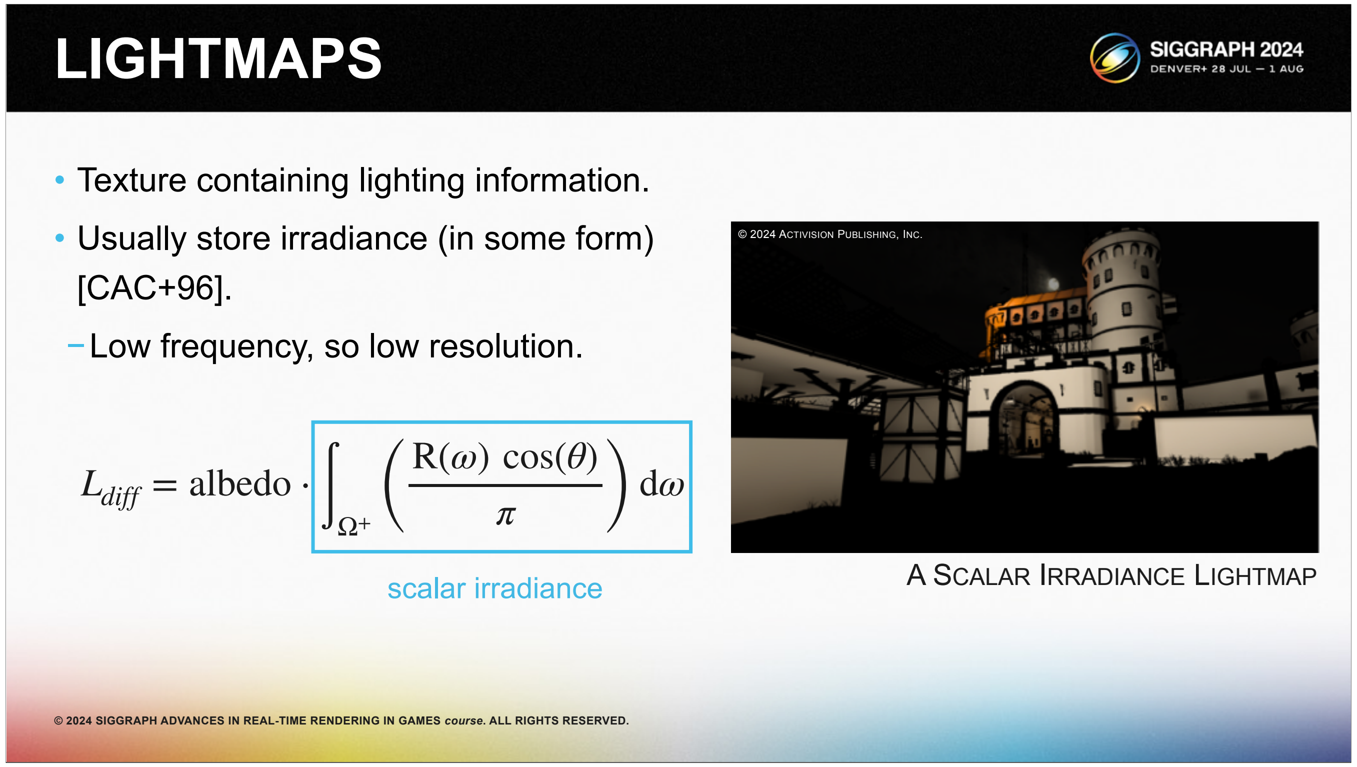
Because they're a linear basis, spherical harmonics have well-behaved blending and interpolation. They're also rotationally invariant, which means projecting a signal and then rotating is the same as rotating and then projecting.
Another particularly useful property is that SH convolution with any radially symmetric function simply becomes a per-band scale. For example, to reconstruct irradiance from an SH representing radiance, you can do that by convolving the radiance SH with a normalised cosine lobe; in practice, this just means that you scale the linear band by 2/3rds and the quadratic band by 1/4 before reconstruction.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

LIGHTMAPS

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Now that we have that grounding, let’s return to lighting. More specifically, let’s take a look at one half of our baked lighting solution: lightmaps.



LIGHTMAPS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Texture containing lighting information.
- Usually store irradiance (in some form) [CAC+96].
  - Low frequency, so low resolution.

$$L_{diff} = \text{albedo} \cdot \underbrace{\boxed{\int_{\Omega^+} \left(\frac{\mathrm{R}(\omega)\ \cos(\theta)}{\pi} \right) \mathrm{d}\omega}}_{\text{scalar irradiance}}$$

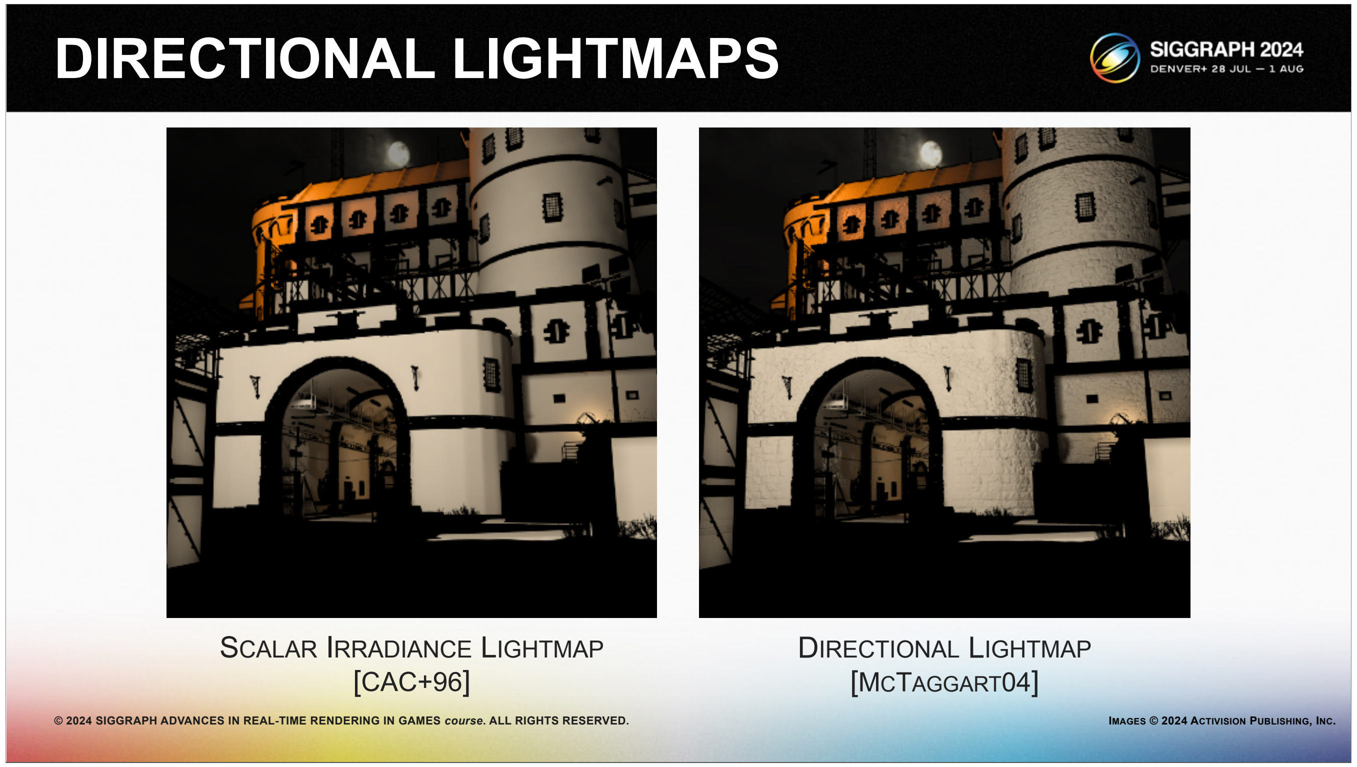
© 2024 Activision Publishing, Inc.

A Scalar Irradiance Lightmap

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Lightmaps are textures containing some lighting information for surfaces in a scene. They've been used for almost thirty years, most commonly to store irradiance. Because irradiance tends to be low frequency, lightmaps can generally be much lower resolution than material textures.

In scalar irradiance lightmaps, each lightmap texel contains the cosine-weighted integral of the radiance over the hemisphere; that gives you a single colour value that you can multiply by the surface albedo to get the reflected irradiance. This allows you to precompute global illumination or bounce lighting for static surfaces and then reconstruct it at runtime by simply sampling from a texture.



DIRECTIONAL LIGHTMAPS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

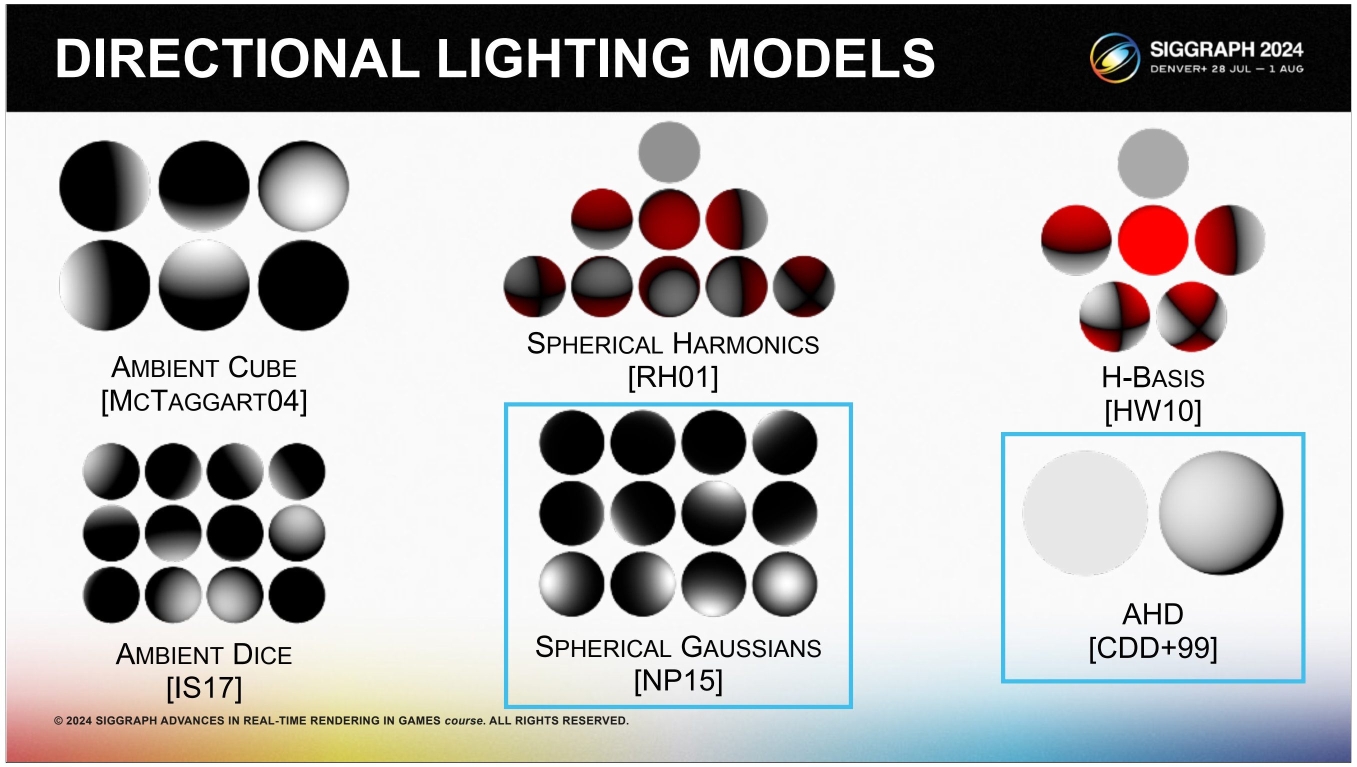
SCALAR IRRADIANCE LIGHTMAP
[CAC+96]

DIRECTIONAL LIGHTMAP
[MCTAGGART04]

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

IMAGES © 2024 ACTIVISION PUBLISHING, INC.

In practice, we often want more detail than what a scalar irradiance lightmap can provide. Real-time rendering makes heavy use of normal maps to add higher frequency detail to coarser geometry. You'd expect a normal facing towards a bright sky to be lighter than one facing a dark wall, so we need to augment our scalar lightmap to encode that variation over the surface hemisphere. These augmented lightmaps are called directional lightmaps, and you use directional lighting models to reconstruct from them.



DIRECTIONAL LIGHTING MODELS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

Ambient Cube
[McTaggart04]

Ambient Dice
[IS17]

Spherical Harmonics
[RH01]

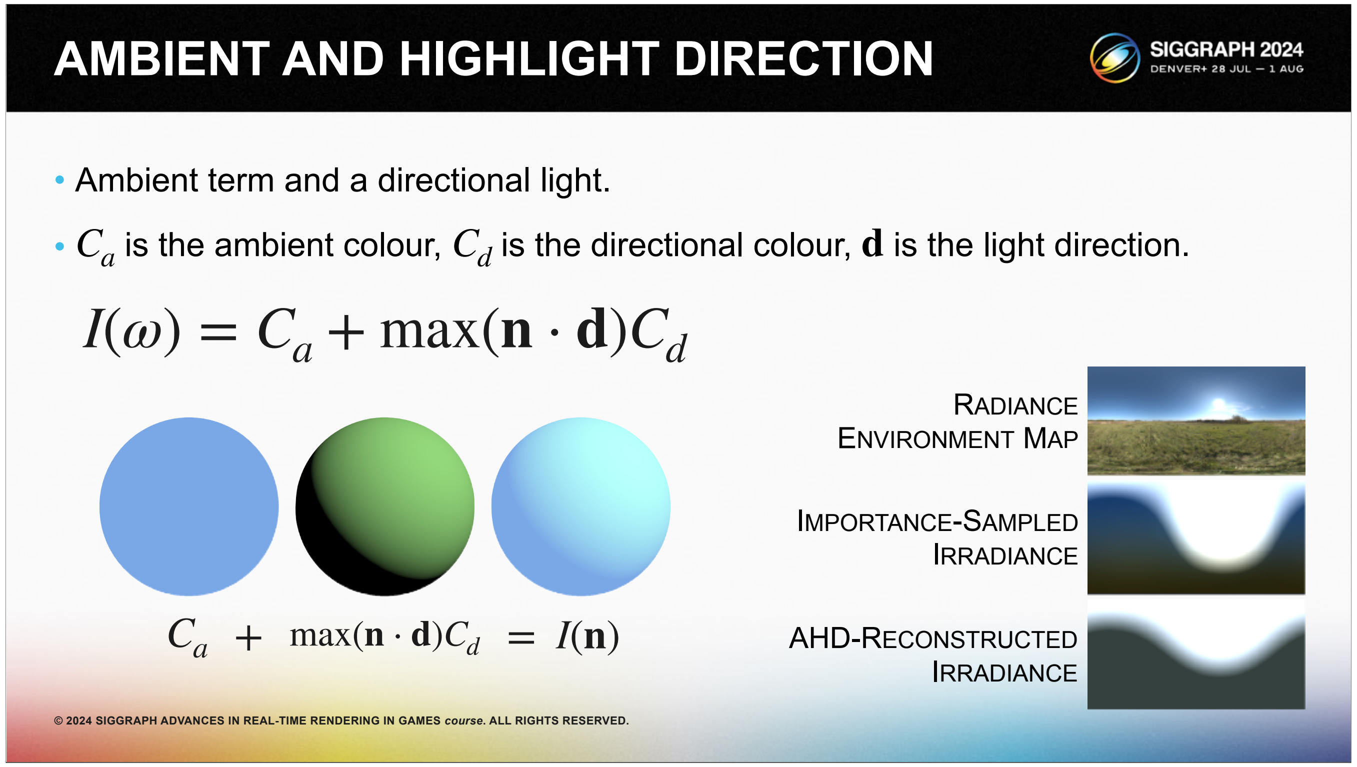
Spherical Gaussians
[NP15]

H-Basis
[HW10]

AHD
[CDD+99]

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

A number of different directional lighting models have been used over the years, which all have different trade-offs; here are just a few of them. One higher quality option is spherical Gaussians, which enable reconstruction of both diffuse and specular lighting; Neubelt and Pettineo presented a talk on how they used them in The Order: 1886 at SIGGRAPH 2015 which is well worth looking at. For our scalability needs, though, we want something more compact, and so we'll start with one of the simplest models you can get: Ambient and Highlight Direction, or AHD.



AMBIENT AND HIGHLIGHT DIRECTION

- Ambient term and a directional light.
- C_a is the ambient colour, C_d is the directional colour, $\mathbf{d}$ is the light direction.

$$I(\omega) = C_a + \max(\mathbf{n} \cdot \mathbf{d})C_d$$

$$C_a \quad + \quad \max(\mathbf{n} \cdot \mathbf{d})C_d \quad = \quad I(\mathbf{n})$$

Radiance Environment Map

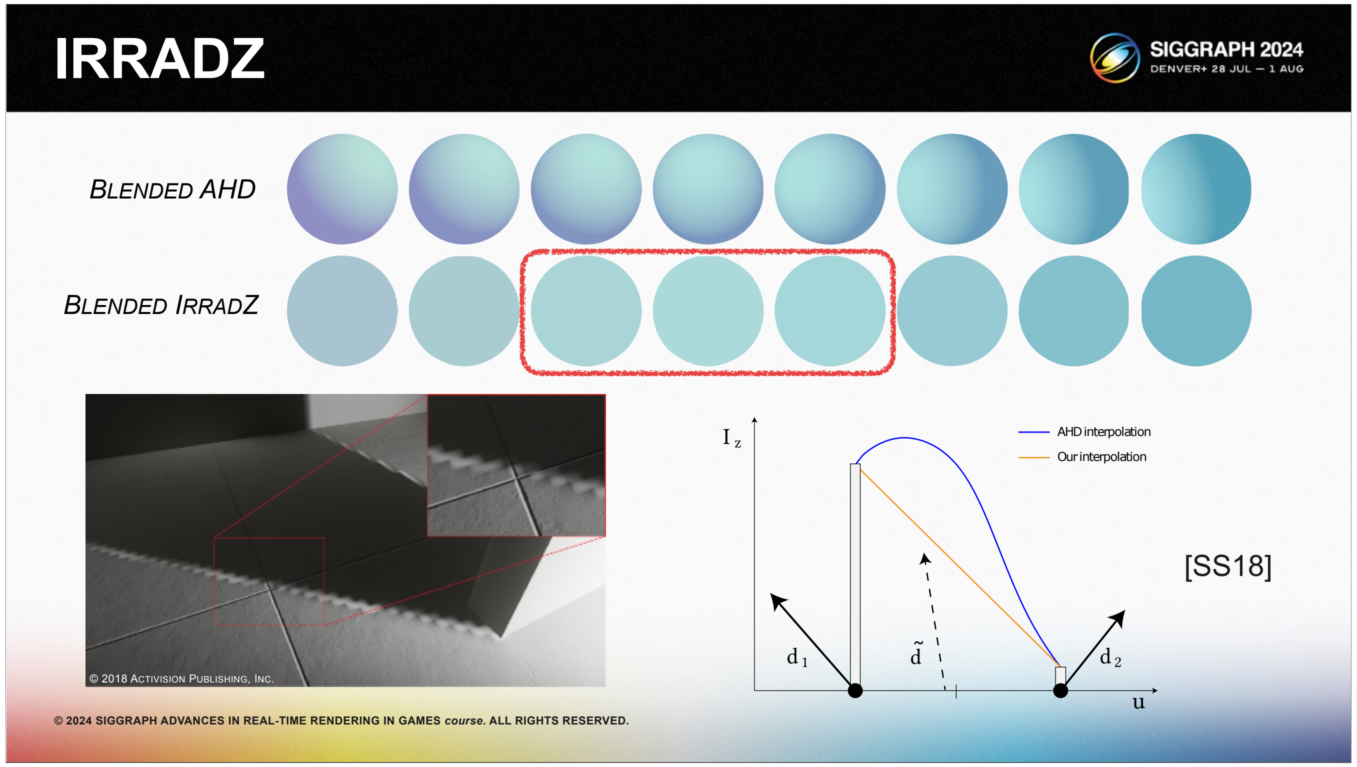
Importance-Sampled Irradiance

AHD-Reconstructed Irradiance

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

AHD decomposes the lighting into an ambient light and a directional light. It's been widely used over the years, from Quake III: Arena to The Last of Us to many of Activision's own titles, and for good reason: it's simple, efficient to evaluate, and provides good quality results.

Image is https://polyhaven.com/a/autumn_field, CC0



IRRADZ

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

BLENDED AHD

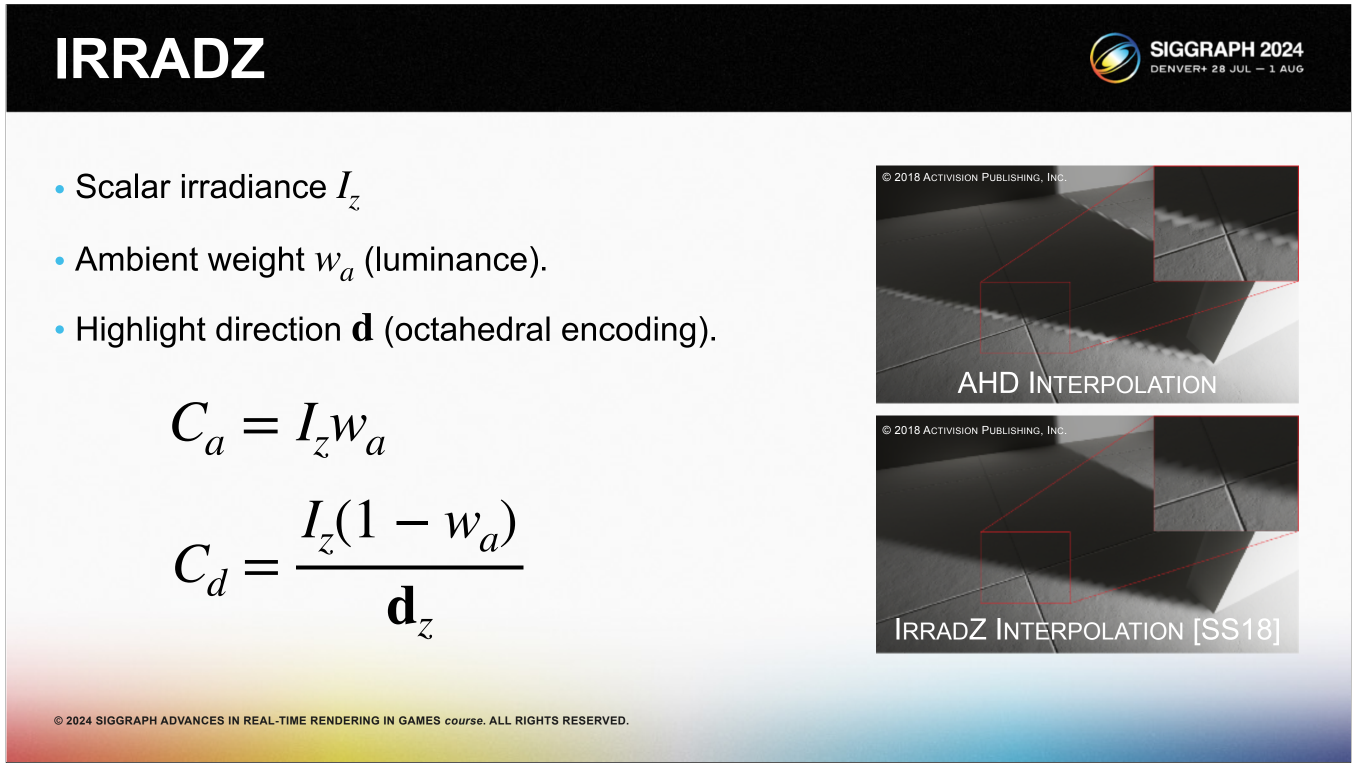
BLENDED IRRADZ

© 2018 ACTIVISION PUBLISHING, INC.

[SS18]

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

It’s not without its problems, though. One major issue is that AHD is a nonlinear lighting model, which means that if you directly store its coefficients in a lightmap, it’s prone to filtering artefacts and overshooting when you blend between texels. You can see here that the blended irradiance in the hemisphere normal, which we call IrradZ, gets brighter than either of the endpoints.



IRRADZ

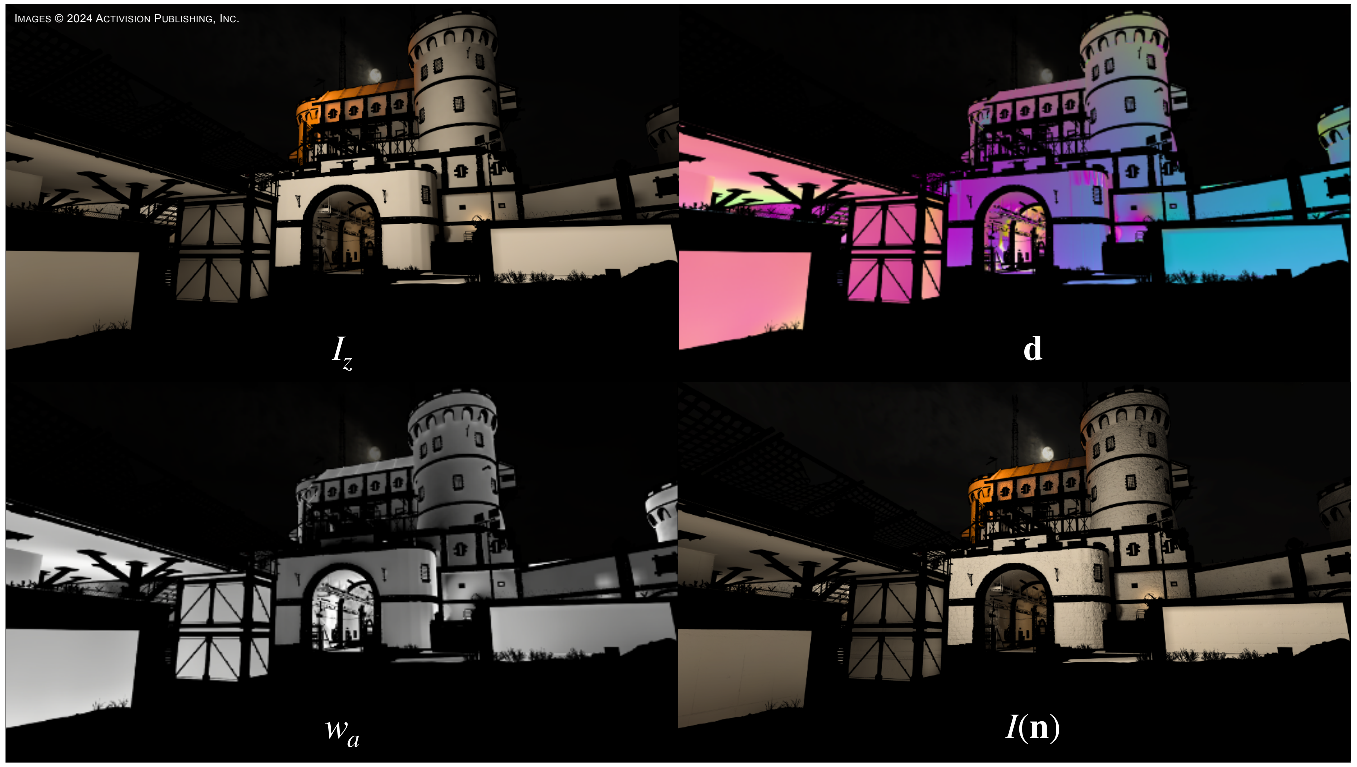
- Scalar irradiance I_z
- Ambient weight w_a (luminance).
- Highlight direction $\mathbf{d}$ (octahedral encoding).

$$C_a = I_z w_a$$

$$C_d = \frac{I_z(1 - w_a)}{\mathbf{d}_z}$$

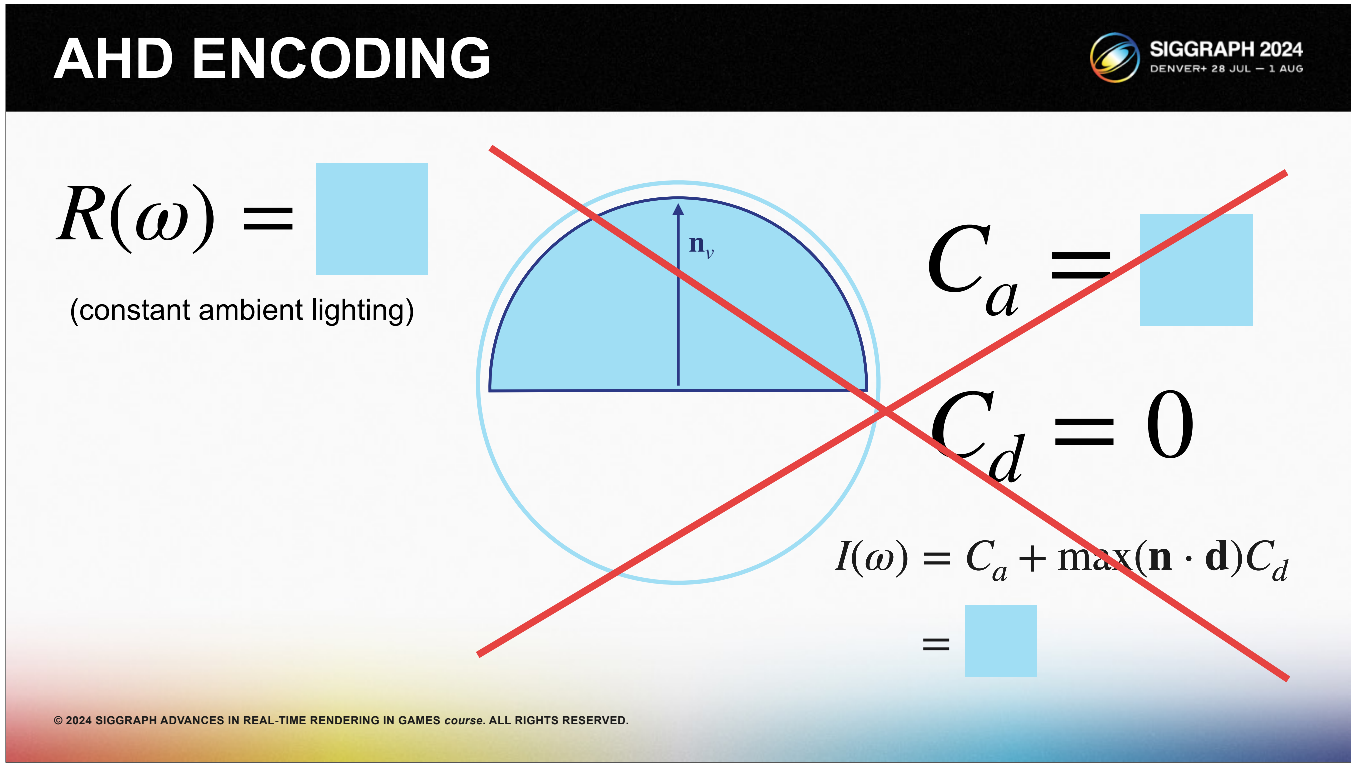
© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

To address this, in 2018 our team published the IrradZ lightmap encoding, where, rather than storing the ambient and directional colour, we store and interpolate the scalar irradiance IrradZ and an ambient weight. We then reconstruct the AHD parameters, keeping Iz invariant in the reconstruction. In addition to fixing the interpolation artefacts, maintaining IrradZ ensures that values at the hemisphere or vertex normal are always exact, and improves accuracy for normals around it.



Images © 2024 Activision Publishing, Inc.

These are the individual IrradZ components that make up our lightmaps, with the lit result in the bottom right. We use block encoding for our textures, so this all takes only 2.5 bytes per texel.



AHD ENCODING

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

$R(\omega) =$ 🟦

(constant ambient lighting)

$C_a =$ 🟦

$C_d = 0$

$I(\omega) = C_a + \max(\mathbf{n} \cdot \mathbf{d}) C_d$

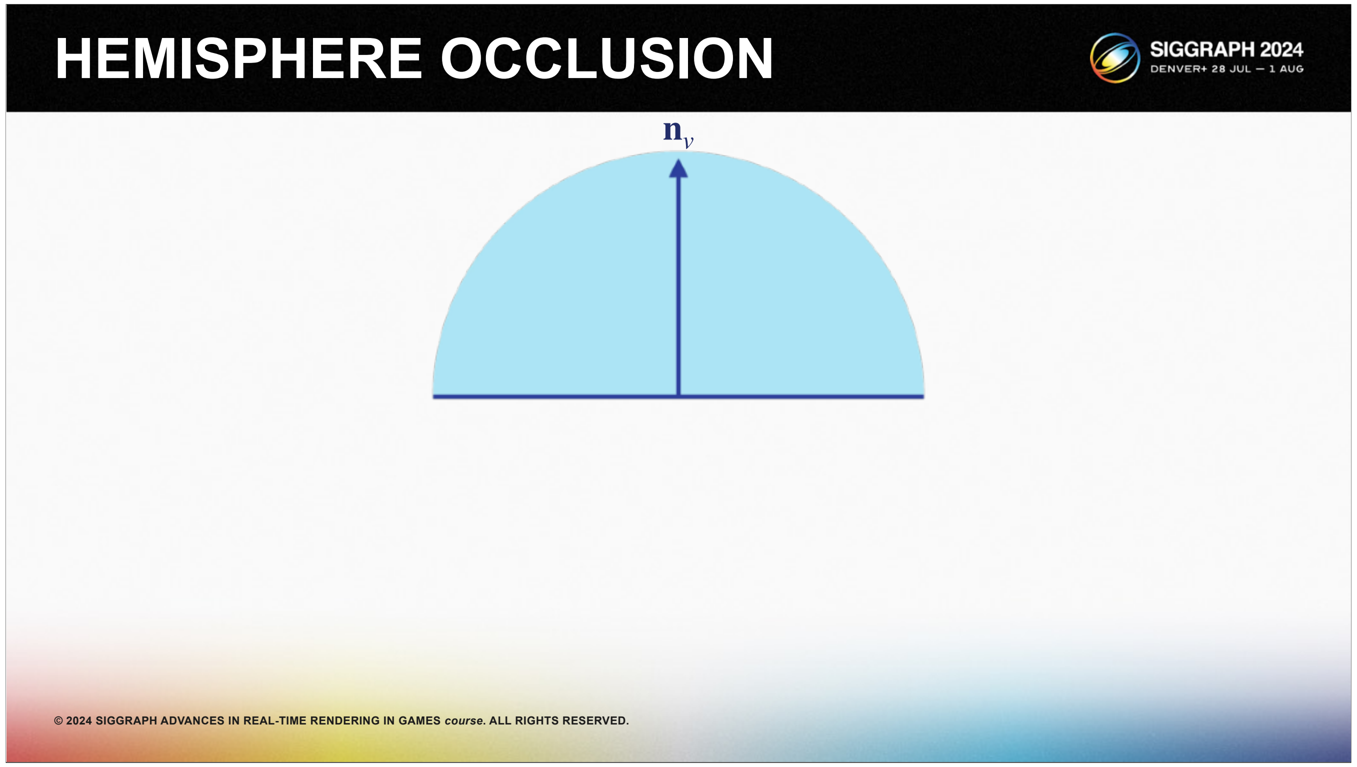
$=$ 🟦

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Okay, so let’s say you’ve decided to use AHD as your runtime lighting model, and you want to encode to it.

Let’s look at how you do that by starting with something simple: constant ambient lighting. What’s the best AHD representation of the irradiance from that lighting?

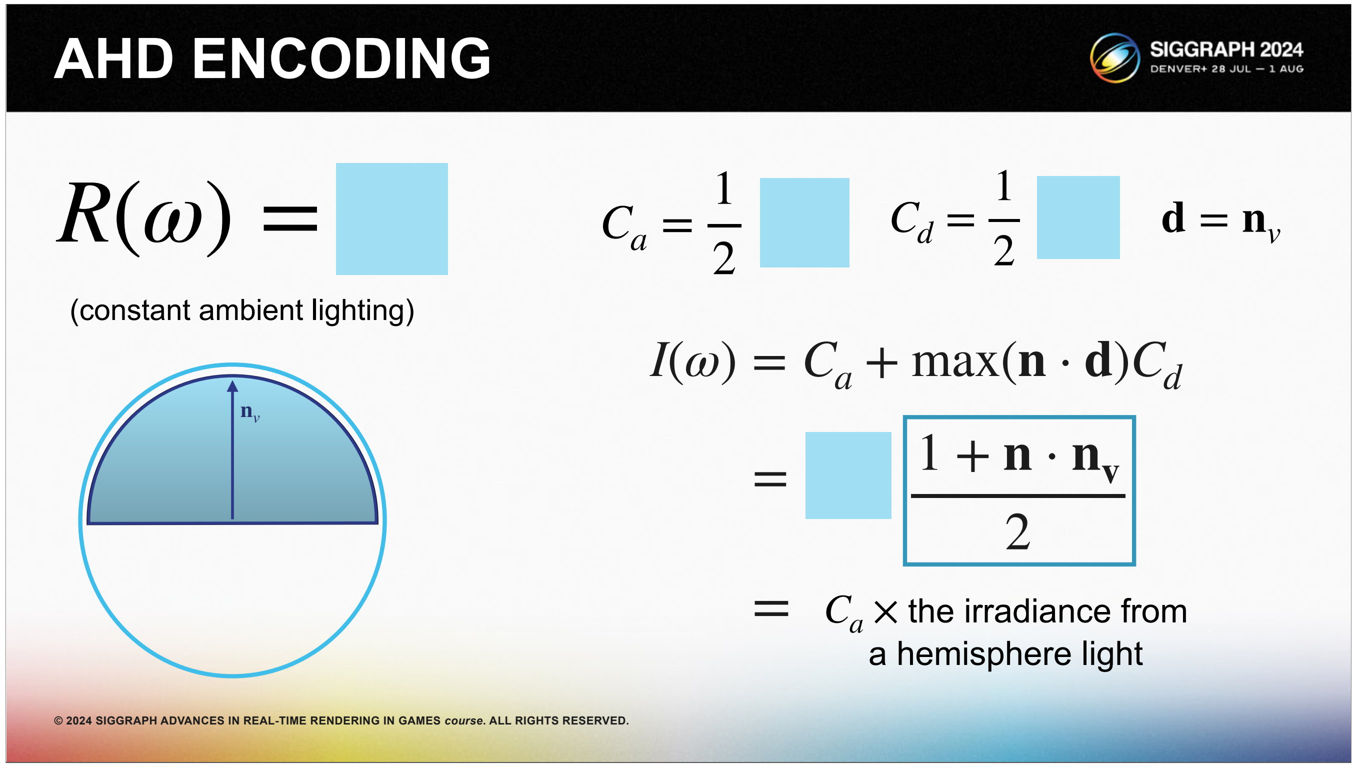
Your first guess might be that it’s straightforward: you set the ambient colour to be the ambient intensity and keep the directional intensity at zero, which gets you your constant colour. In reality, though, that’s not what the irradiance looks like.



HEMISPHERE OCCLUSION

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

In practice, as normals face away from the hemisphere normal, more and more of the incoming light gets blocked by the surface. This is called *hemispherical occlusion,* and it produces a gradient of irradiance that gets darker towards the edges. If you solve for the AHD that best represents this irradiance,



AHD ENCODING

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

$R(\omega) =$ 🟦

(constant ambient lighting)

$C_a = \frac{1}{2}$ 🟦 $C_d = \frac{1}{2}$ 🟦 $\mathbf{d} = \mathbf{n}_v$

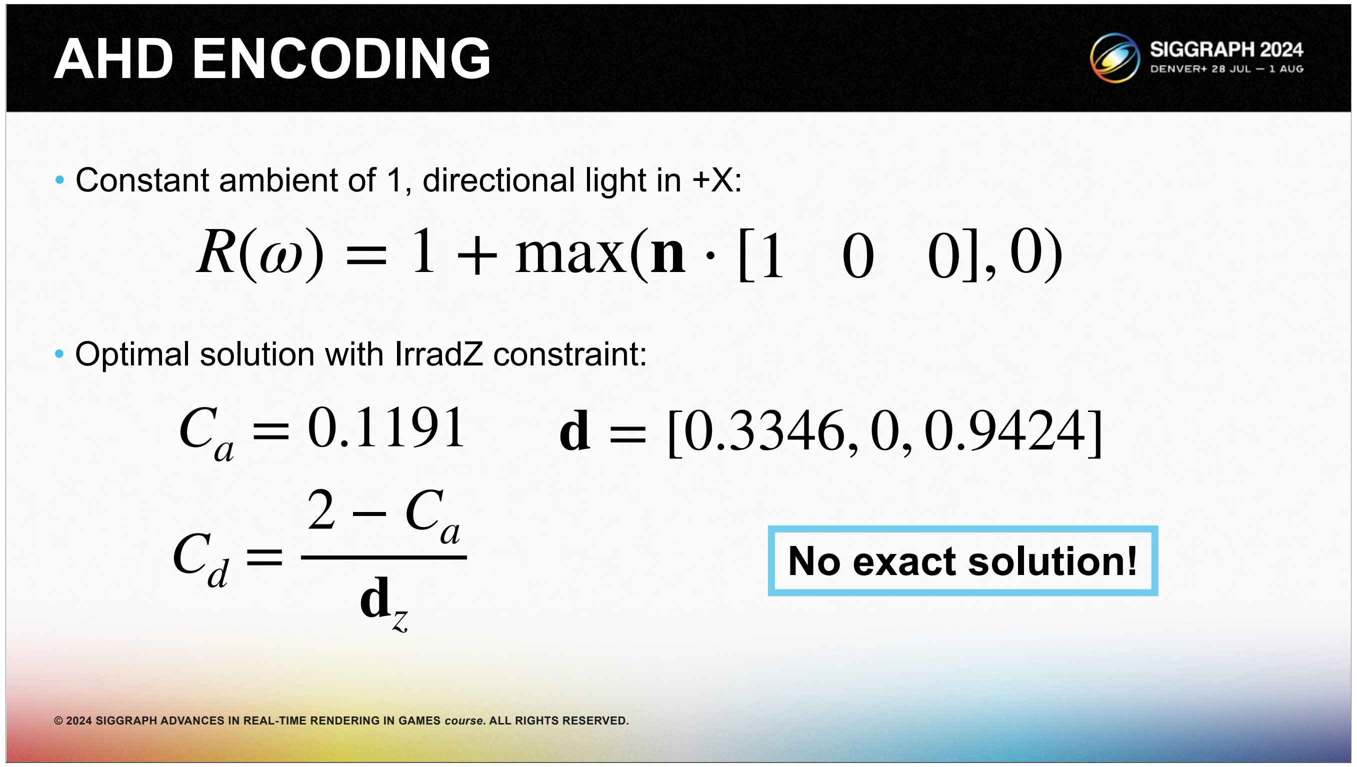
$$I(\omega) = C_a + \max(\mathbf{n} \cdot \mathbf{d}) C_d$$

$= $ 🟦 $\boxed{\frac{1 + \mathbf{n} \cdot \mathbf{n_v}}{2}}$

$= C_a \times$ the irradiance from a hemisphere light

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

You’re left with this encoding; C_a is half the ambient colour, C_d is also half the ambient colour, and the highlight direction is oriented with the vertex normal. This turns out to be exactly the equation for a hemisphere light!



AHD ENCODING

- Constant ambient of 1, directional light in +X:

$$R(\omega) = 1 + \max(\mathbf{n} \cdot [1 \quad 0 \quad 0], 0)$$

- Optimal solution with IrradZ constraint:

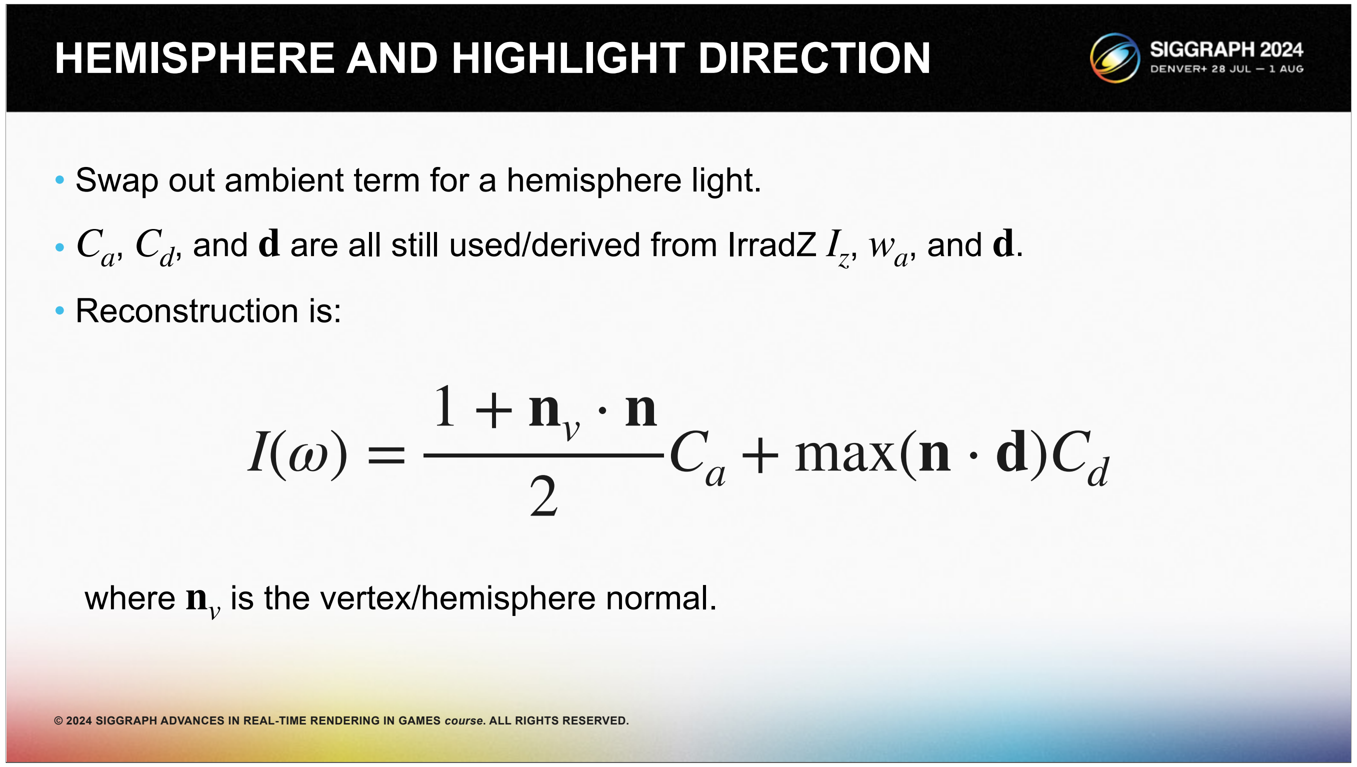
$$C_a = 0.1191 \qquad \mathbf{d} = [0.3346, 0, 0.9424]$$

$$C_d = \frac{2 - C_a}{\mathbf{d}_z}$$

No exact solution!

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

If we try adding a directional light pointing off to the side, we get an inexact result. We can’t exactly represent the irradiance from an ambient and a directional light with the AHD model!



HEMISPHERE AND HIGHLIGHT DIRECTION

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Swap out ambient term for a hemisphere light.
- C_a, C_d, and $\mathbf{d}$ are all still used/derived from IrradZ I_z, w_a, and $\mathbf{d}$.
- Reconstruction is:

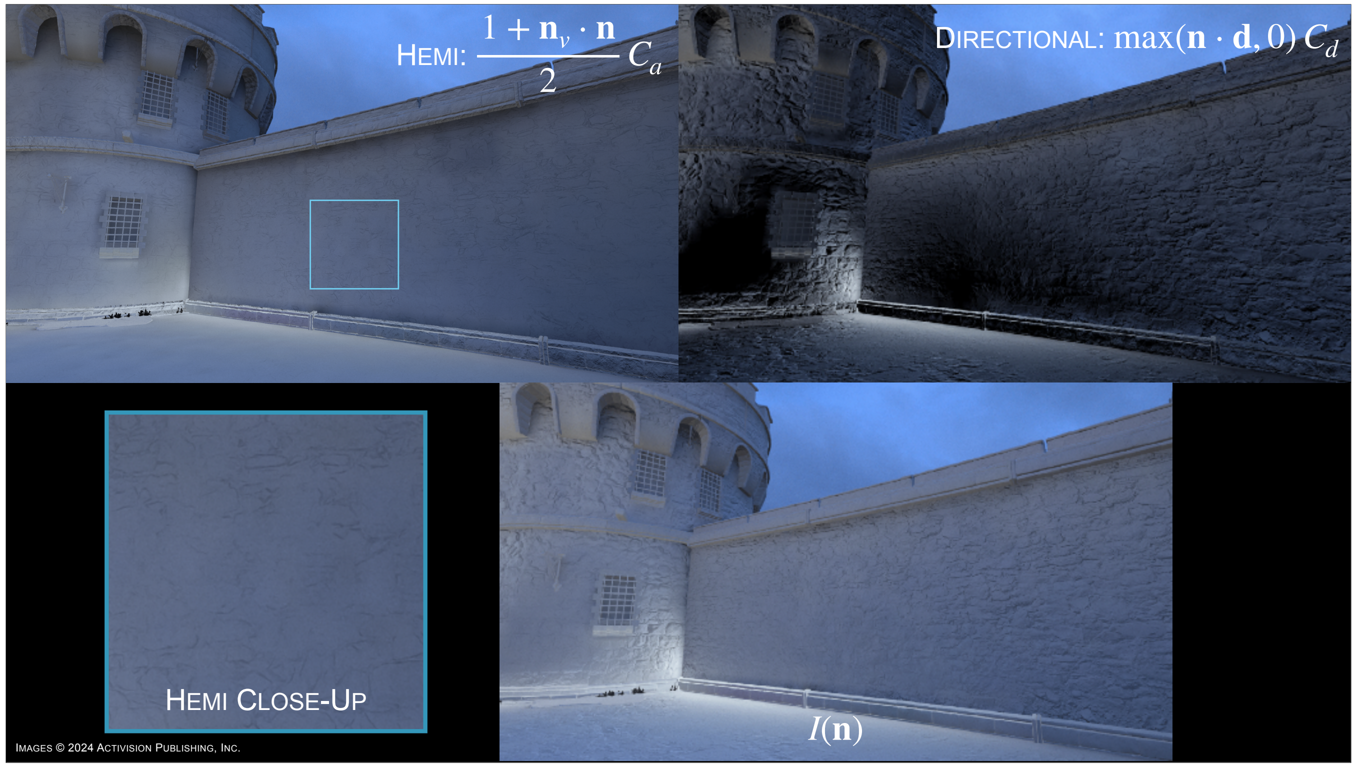
$$I(\omega) = \frac{1 + \mathbf{n}_v \cdot \mathbf{n}}{2} C_a + \max(\mathbf{n} \cdot \mathbf{d}) C_d$$

where $\mathbf{n}_v$ is the vertex/hemisphere normal.

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

This was the motivation for us to replace the ambient light in the reconstruction with a hemisphere light, producing what we call HHD, or Hemisphere and Highlight Direction. We can reuse the same IrradZ encoding and coefficients, and reconstruct using a hemisphere light to scale the ambient colour.

Now the encoding of the irradiance from a constant ambient term plus a directional light is trivial and exact.



HEMI: $\frac{1 + \mathbf{n}_y \cdot \mathbf{n}}{2} C_a$

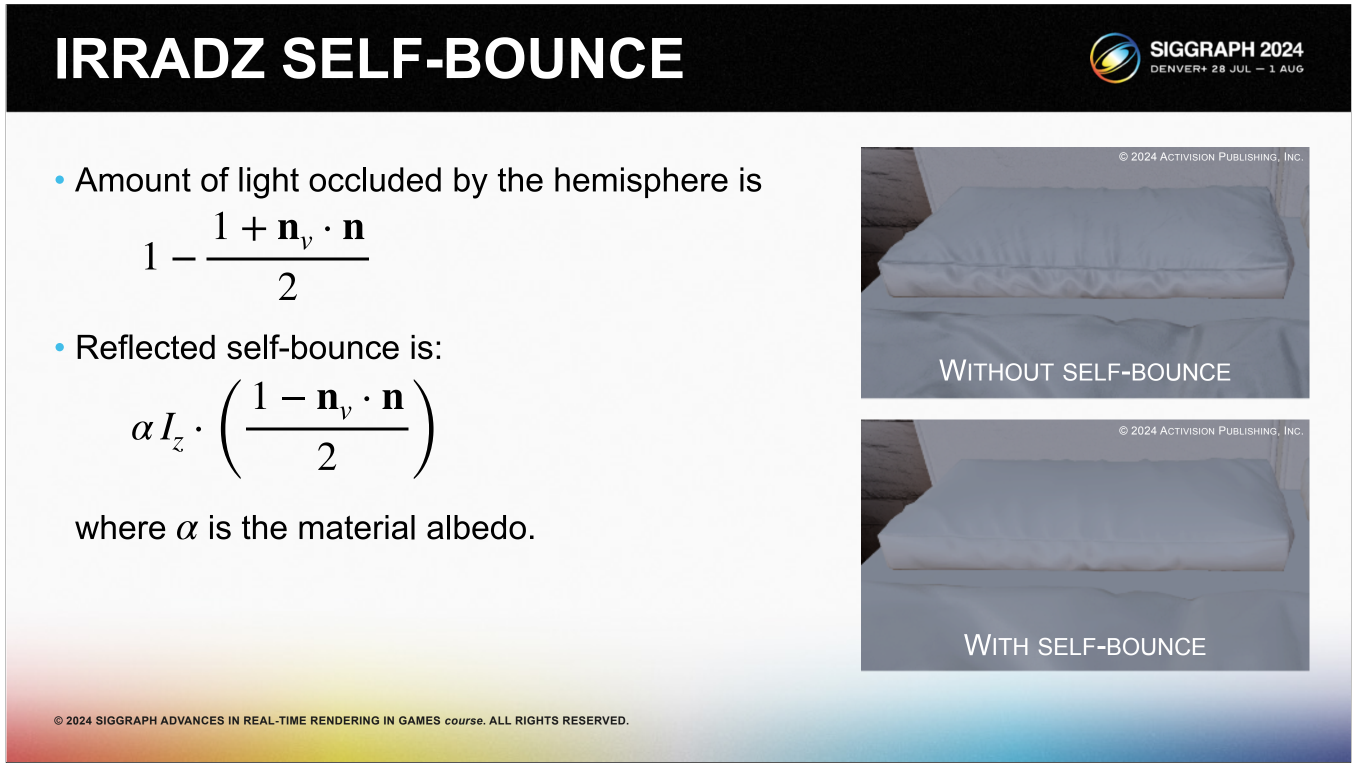
DIRECTIONAL: $\max(\mathbf{n} \cdot \mathbf{d}, 0)\, C_d$

HEMI CLOSE-UP

$I(\mathbf{n})$

IMAGES © 2024 ACTIVISION PUBLISHING, INC.

These are the components of our reconstructed lighting. The contribution from the hemisphere light is in the top left; if you look closely you can see some subtle directional variation from the surface occlusion. The directional is in the top right, and they're summed together to produce our final combined irradiance.



IRRADZ SELF-BOUNCE

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Amount of light occluded by the hemisphere is

$$1 - \frac{1 + \mathbf{n}_v \cdot \mathbf{n}}{2}$$

- Reflected self-bounce is:

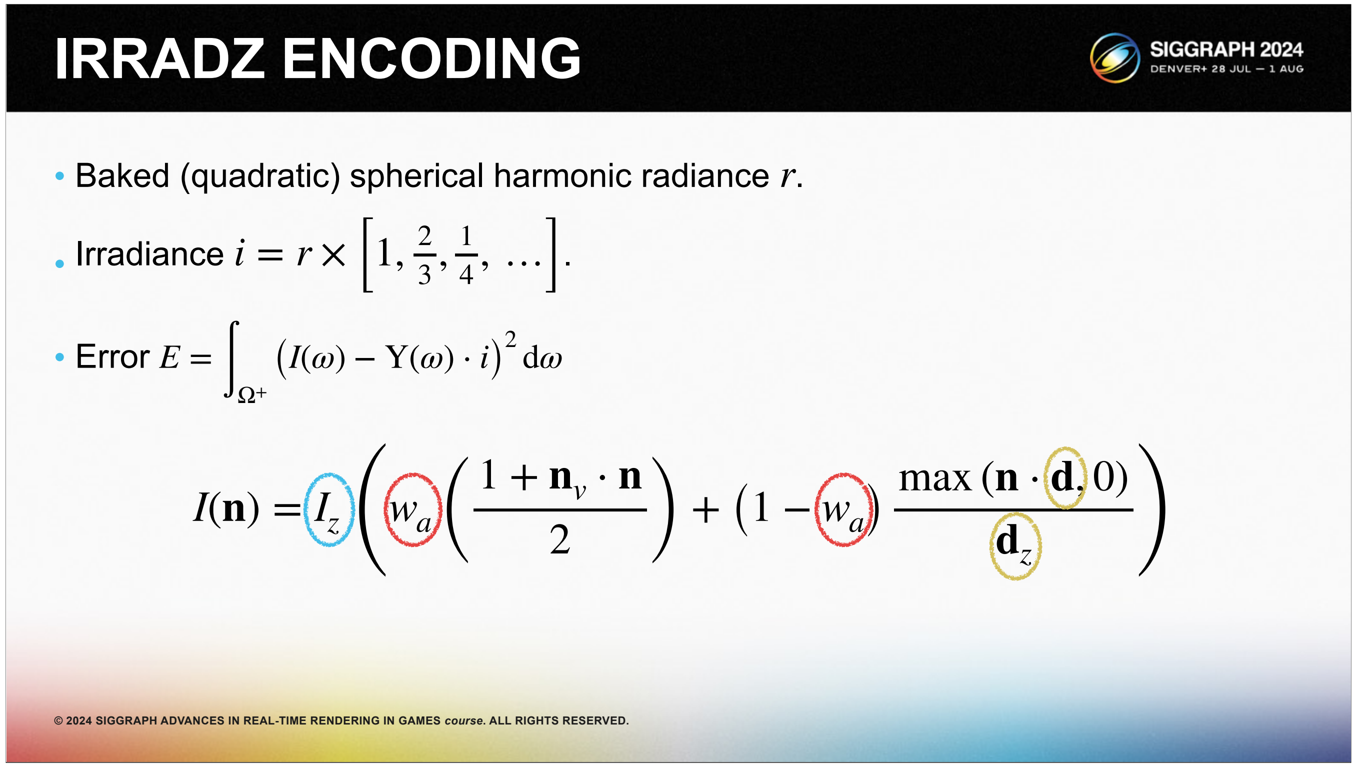
$$\alpha I_z \cdot \left(\frac{1 - \mathbf{n}_v \cdot \mathbf{n}}{2}\right)$$

where α is the material albedo.

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

We can also use hemisphere lights to model surface self-bounce. If the irradiance from the visible hemisphere is given by a hemisphere light, the irradiance from the negative hemisphere must be one minus that. We can approximate the reflected irradiance to be the incident radiance, IrradZ, times the surface albedo.

Conveniently, we already have the IrradZ value from our lightmap, so this is a very cheap adjustment that can make a big difference for higher-albedo materials. Enlighten’s radiosity system from the early 2010s used this same self-bounce trick.



IRRADZ ENCODING

- Baked (quadratic) spherical harmonic radiance r.
- Irradiance $i = r \times \left[1, \frac{2}{3}, \frac{1}{4}, \ldots\right]$.
- Error $E = \int_{\Omega^+} \left(I(\omega) - \mathrm{Y}(\omega) \cdot i\right)^2 \mathrm{d}\omega$

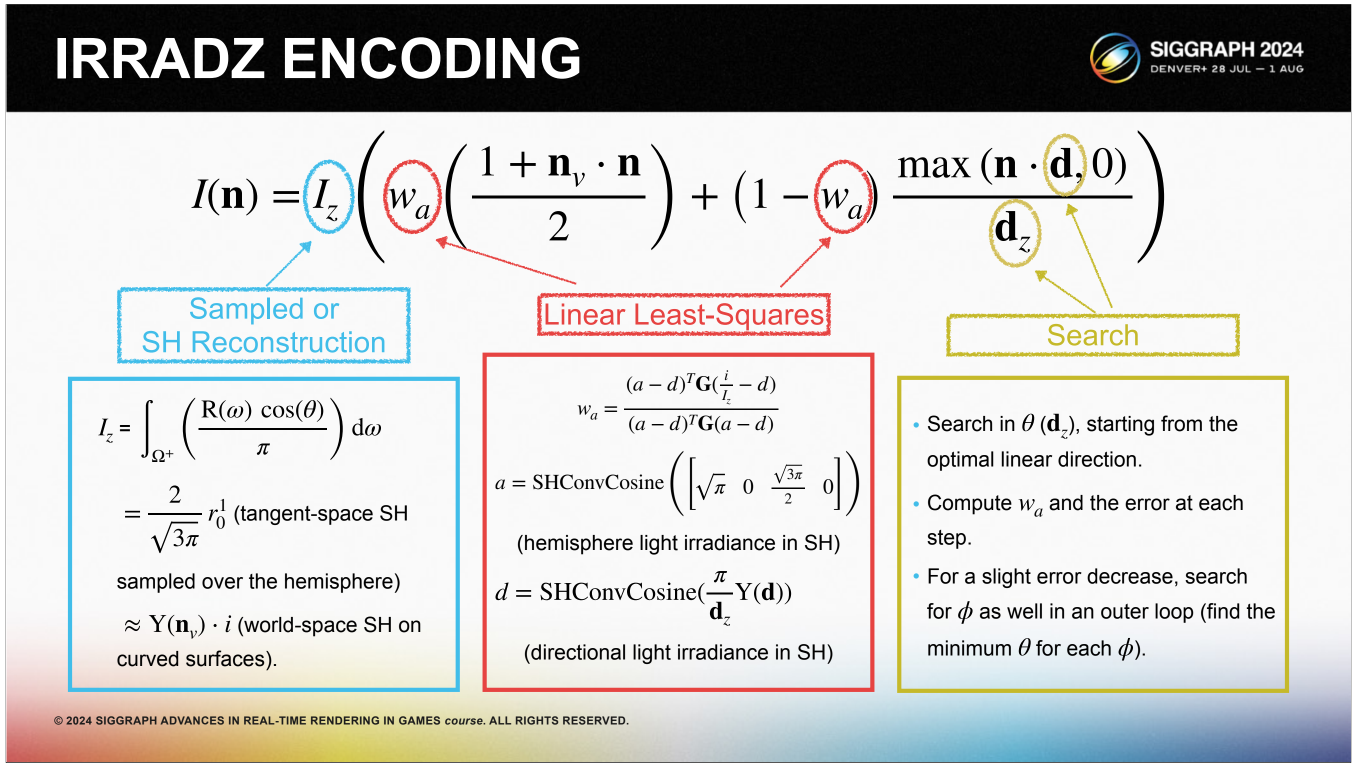
$$I(\mathbf{n}) = I_z \left(w_a \left(\frac{1 + \mathbf{n}_v \cdot \mathbf{n}}{2} \right) + \left(1 - w_a\right) \frac{\max\left(\mathbf{n} \cdot \mathbf{d}, 0\right)}{\mathbf{d}_z} \right)$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Now that we have our lighting model, it's time to revisit how to encode it.

In practice, our lighting environments are a bit more complicated than a constant ambient term plus a directional light. During the lightmap bake, we have radiance over at least a hemisphere stored as spherical harmonics, and we want to find the best approximation of the irradiance of that as HHD. Our error is the difference between the reconstructed irradiance and the SH irradiance.

If you expand out the reconstruction function, we have three parameters we want to find. We have the irradiance in the hemisphere normal Iz, the ambient weight w_a, which linearly blends between the hemisphere light and the directional light, and the highlight direction d. We can use a different strategy for each.



IRRADZ ENCODING

$$I(\mathbf{n}) = I_z\left(w_a\left(\frac{1+\mathbf{n}_v\cdot\mathbf{n}}{2}\right) + (1-w_a)\,\frac{\max\left(\mathbf{n}\cdot\mathbf{d},0\right)}{\mathbf{d}_z}\right)$$

Sampled or
SH Reconstruction

Linear Least-Squares

Search

$$I_z = \int_{\Omega^+}\left(\frac{\mathrm{R}(\omega)\ \cos(\theta)}{\pi}\right)\mathrm{d}\omega$$

$= \frac{2}{\sqrt{3\pi}} r_0^1$ (tangent-space SH sampled over the hemisphere)

$\approx \mathrm{Y}(\mathbf{n}_v)\cdot i$ (world-space SH on curved surfaces).

$$w_a = \frac{(a-d)^T\mathbf{G}(\frac{i}{I_z}-d)}{(a-d)^T\mathbf{G}(a-d)}$$

$$a = \mathrm{SHConvCosine}\left(\begin{bmatrix}\sqrt{\pi} & 0 & \frac{\sqrt{3\pi}}{2} & 0\end{bmatrix}\right)$$

(hemisphere light irradiance in SH)

$$d = \mathrm{SHConvCosine}(\frac{\pi}{\mathbf{d}_z}\mathrm{Y}(\mathbf{d}))$$

(directional light irradiance in SH)

- Search in θ ($\mathbf{d}_z$), starting from the optimal linear direction.
- Compute w_a and the error at each step.
- For a slight error decrease, search for ϕ as well in an outer loop (find the minimum θ for each ϕ).

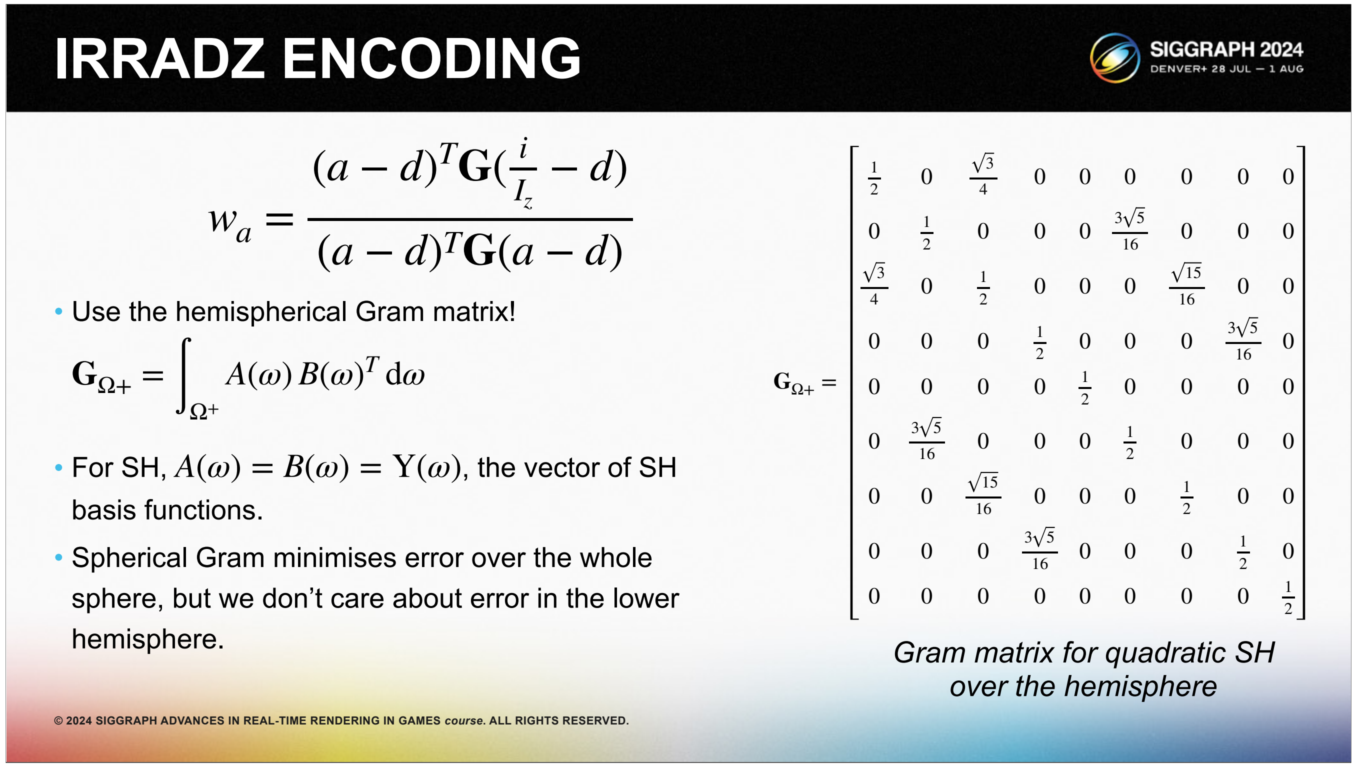
© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

We can directly sample I_z in the bake or approximate it by reconstruction from the irradiance SH.

The ambient weight is a linear parameter, so we can find it using linear least-squares.

The highlight direction is a nonlinear parameter, so we need to search for it. Because irradiance is low-frequency and the error function is well-behaved, we can perform a golden section search in theta, computing w_a and the error at each step.

There are a couple of subtleties here that are worth drawing attention to.



IRRADZ ENCODING

$$w_a = \frac{(a-d)^T \mathbf{G}(\frac{i}{I_z} - d)}{(a-d)^T \mathbf{G}(a-d)}$$

- Use the hemispherical Gram matrix!

$$\mathbf{G}_{\Omega+} = \int_{\Omega+} A(\omega)\, B(\omega)^T \, \mathrm{d}\omega$$

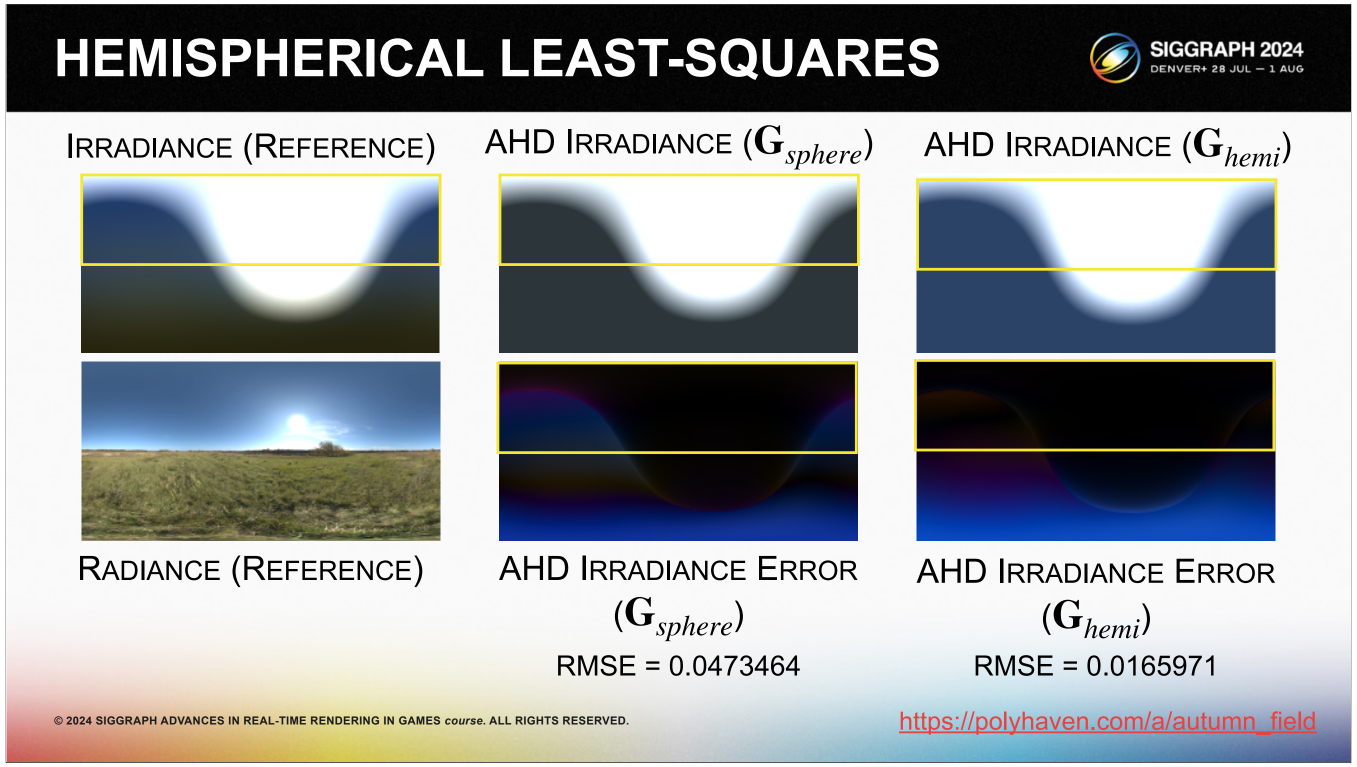
- For SH, $A(\omega) = B(\omega) = \mathrm{Y}(\omega)$, the vector of SH basis functions.
- Spherical Gram minimises error over the whole sphere, but we don't care about error in the lower hemisphere.

$$\mathbf{G}_{\Omega+} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{\sqrt{3}}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 \\ \frac{\sqrt{3}}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{\sqrt{15}}{16} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{15}}{16} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

Gram matrix for quadratic SH over the hemisphere

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

First is that for the ambient weight, it's important we use the hemispherical Gram matrix, which you compute by integrating the product of SH basis functions over the *hemisphere*. If you use the spherical Gram matrix, you're minimising error over the whole sphere, which wastes quality on something you're never going to see.



HEMISPHERICAL LEAST-SQUARES

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

Irradiance (Reference)

AHD Irradiance ($\mathbf{G}_{sphere}$)

AHD Irradiance ($\mathbf{G}_{hemi}$)

Radiance (Reference)

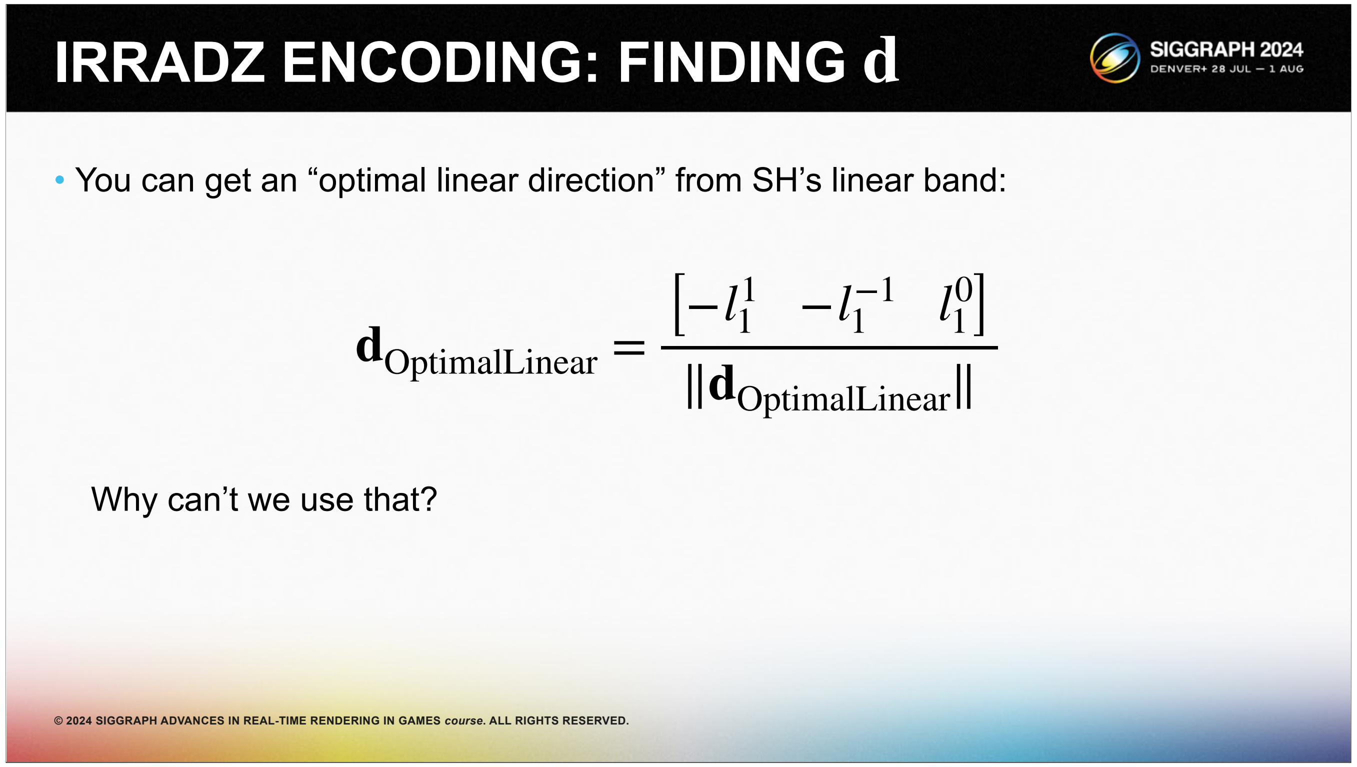
AHD Irradiance Error ($\mathbf{G}_{sphere}$)
RMSE = 0.0473464

AHD Irradiance Error ($\mathbf{G}_{hemi}$)
RMSE = 0.0165971

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

https://polyhaven.com/a/autumn_field

Put simply, if you’re only looking up, the fact that the ground is green doesn’t matter to you, and you don’t want that in your ambient colour. You only need to capture that the sky is blue.



IRRADZ ENCODING: FINDING $\mathbf{d}$

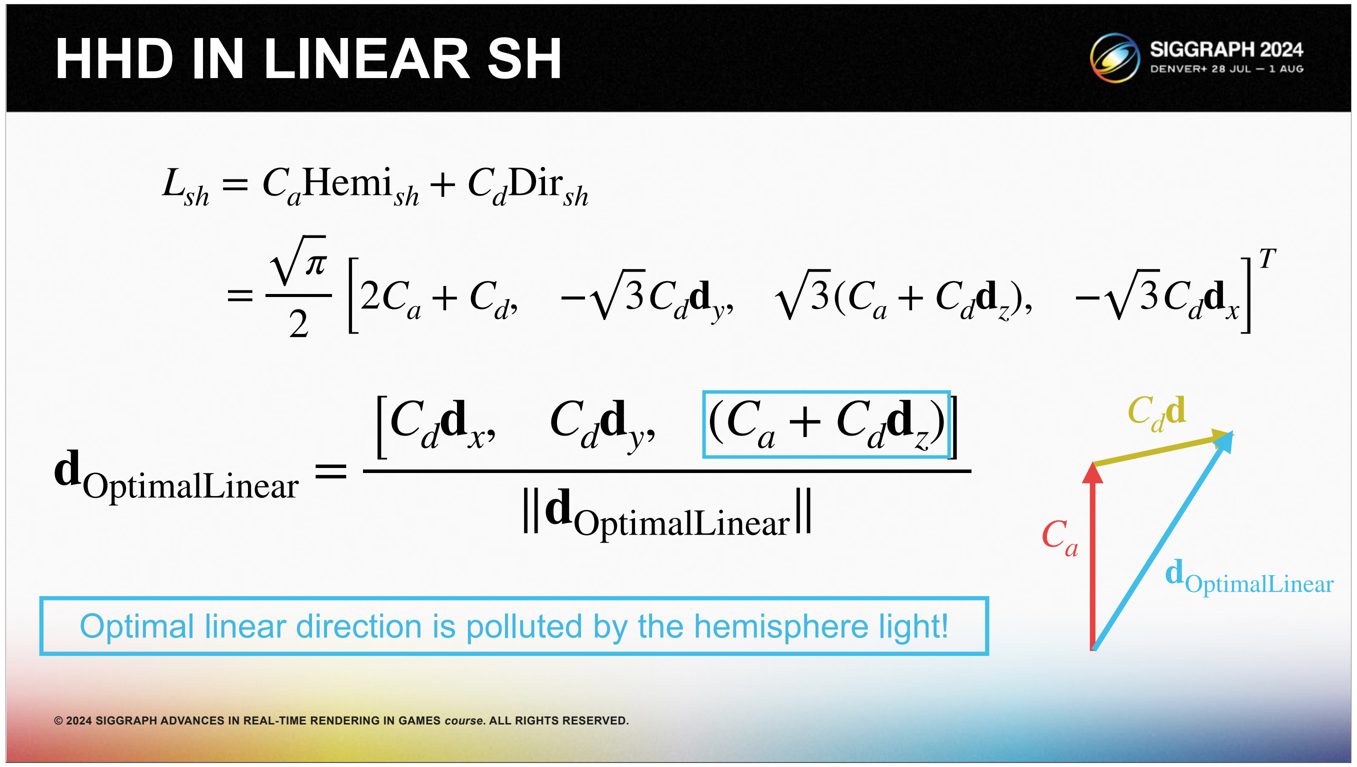
- You can get an “optimal linear direction” from SH’s linear band:

$$\mathbf{d}_{\text{OptimalLinear}} = \frac{\begin{bmatrix} -l_1^1 & -l_1^{-1} & l_1^0 \end{bmatrix}}{\|\mathbf{d}_{\text{OptimalLinear}}\|}$$

Why can’t we use that?

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

The second subtlety is in why we need to search for the highlight direction. In the past, games have used what’s known as the “optimal linear direction”, which you can get from spherical harmonics by normalising and swapping the components in the linear band. Why can’t we use that?



HHD IN LINEAR SH

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

$$L_{sh} = C_a \mathrm{Hemi}_{sh} + C_d \mathrm{Dir}_{sh}$$

$$= \frac{\sqrt{\pi}}{2} \left[2C_a + C_d, \quad -\sqrt{3} C_d \mathbf{d}_y, \quad \sqrt{3}(C_a + C_d \mathbf{d}_z), \quad -\sqrt{3} C_d \mathbf{d}_x \right]^T$$

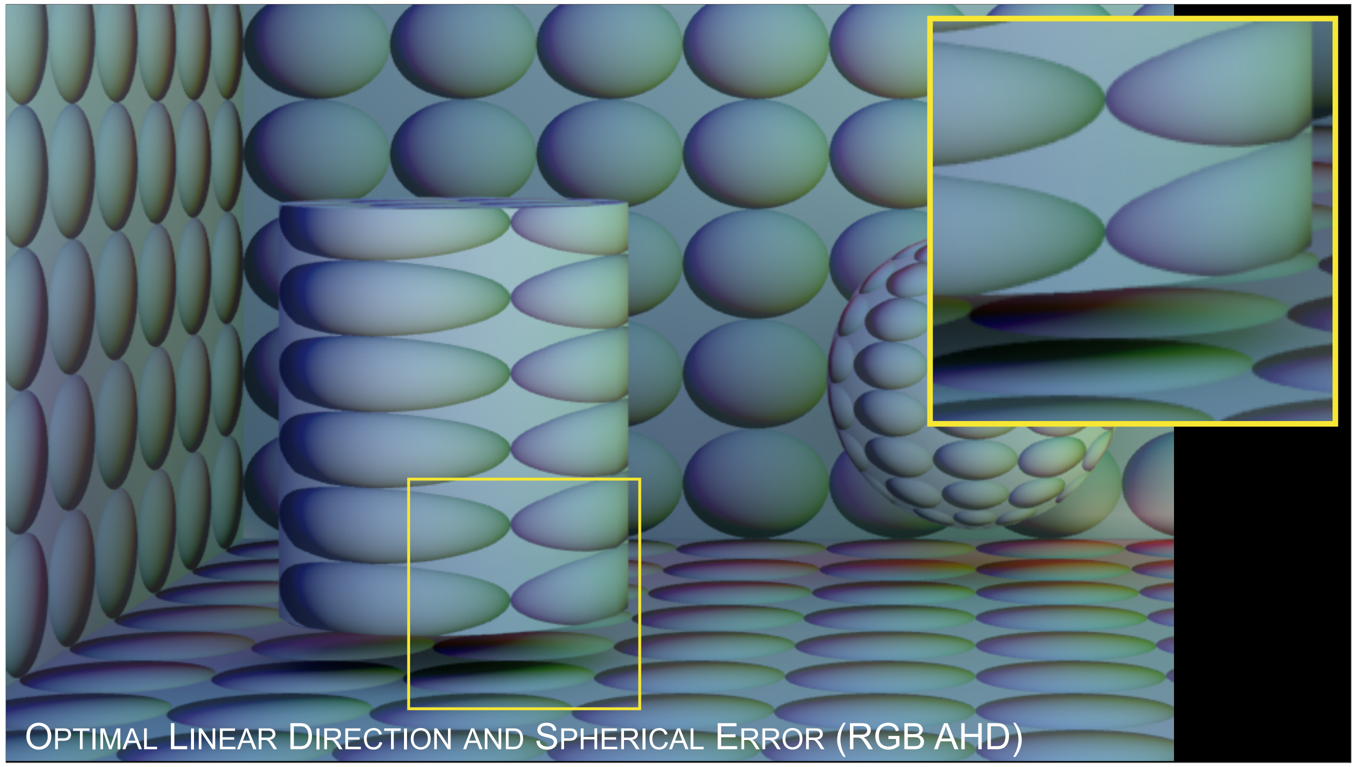
$$\mathbf{d}_{\mathrm{OptimalLinear}} = \frac{\left[C_d \mathbf{d}_x, \quad C_d \mathbf{d}_y, \quad \boxed{(C_a + C_d \mathbf{d}_z)} \right]}{\|\mathbf{d}_{\mathrm{OptimalLinear}}\|}$$

Optimal linear direction is polluted by the hemisphere light!

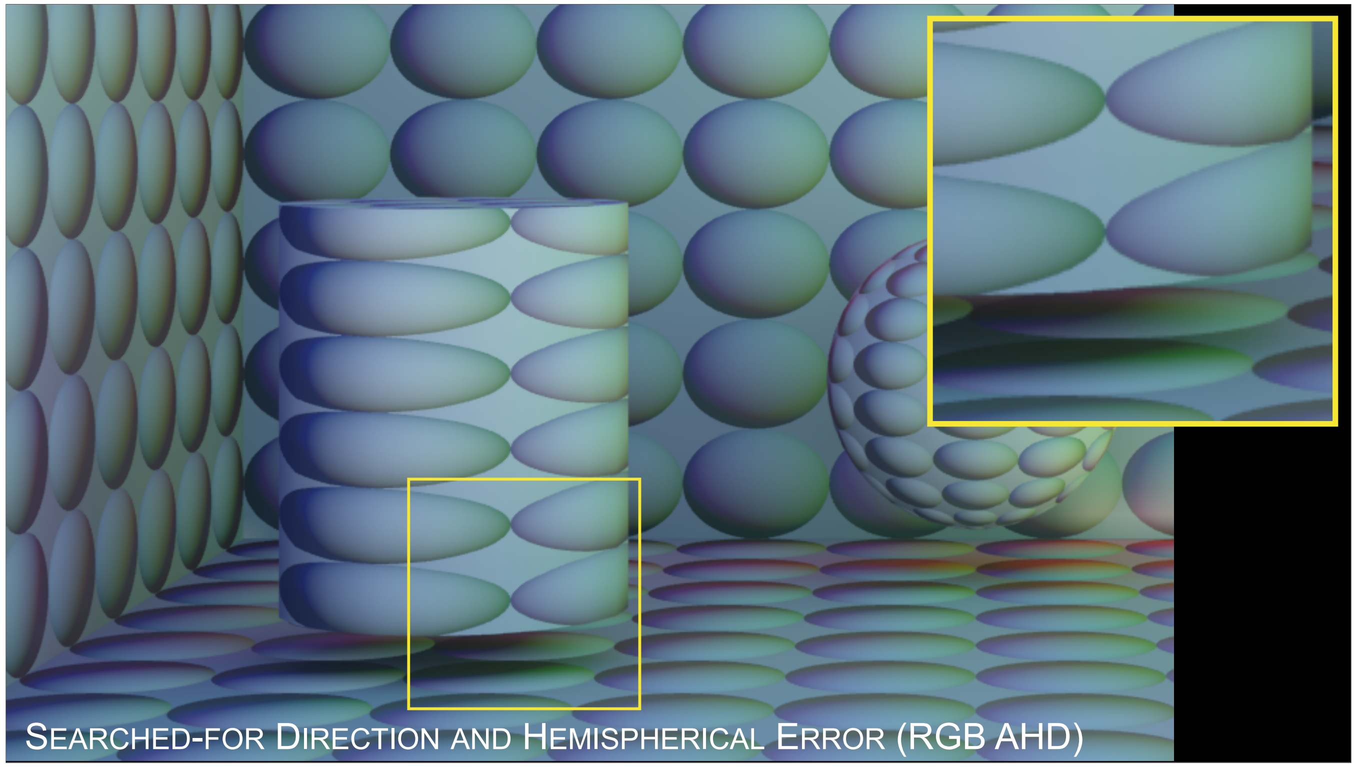
© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Well, let’s look at the projection of HHD into spherical harmonics. If we then take the optimal linear direction from the linear band, we can see that the Z component is polluted by the hemisphere light, so the highlight direction is shifted towards the hemisphere normal. This means our directional light won’t be tilted enough to the side, it won’t round-trip, and we’ll get higher error.

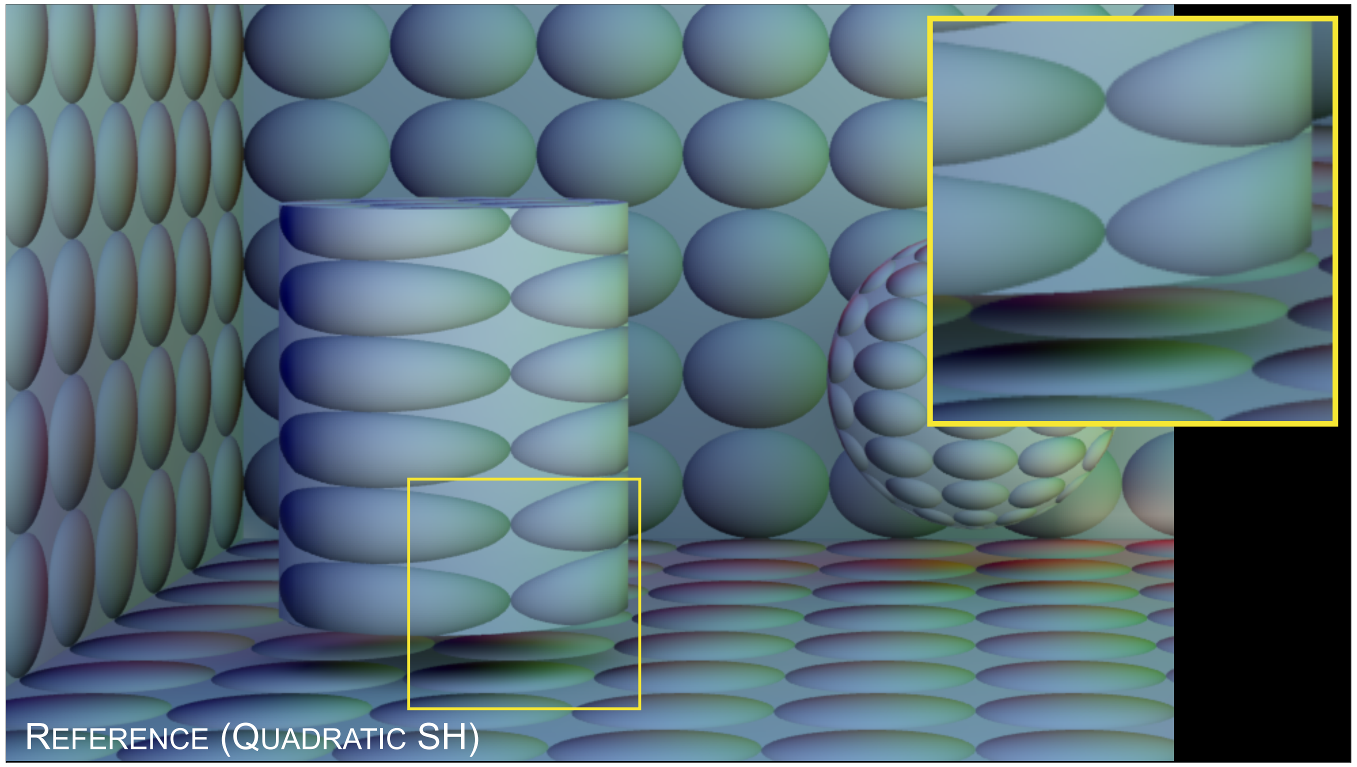
In a more complex example, you could also have energy in higher order spherical harmonic bands that influences the highlight direction.



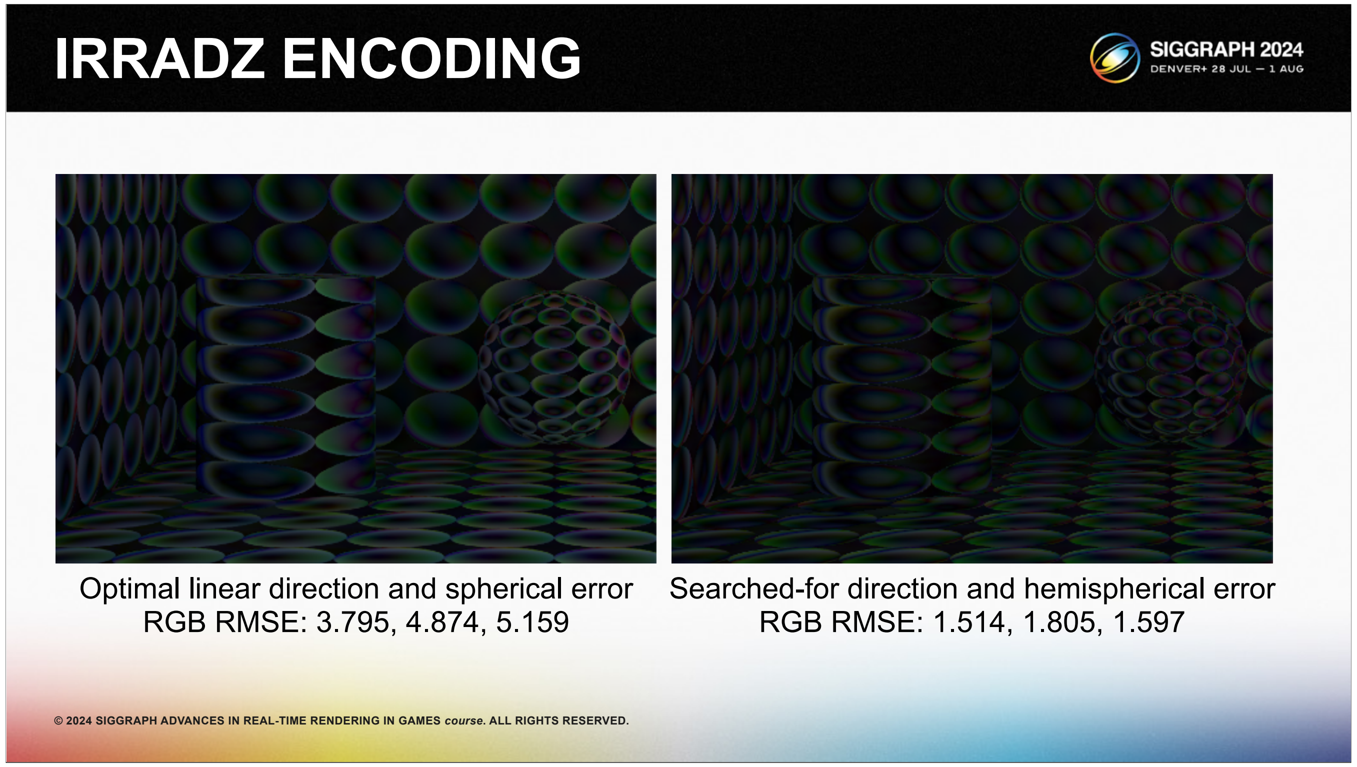
To compare visually, let’s start from using the optimal linear direction and spherical Gram matrix; this is what a lot of games, including ours, have shipped with in the past. Keep a close watch on the left edges of these spheres here.



Using the searched-for direction and hemispherical error brightens up the edges, which, compared to the reference…



… is a much closer match.



IRRADZ ENCODING

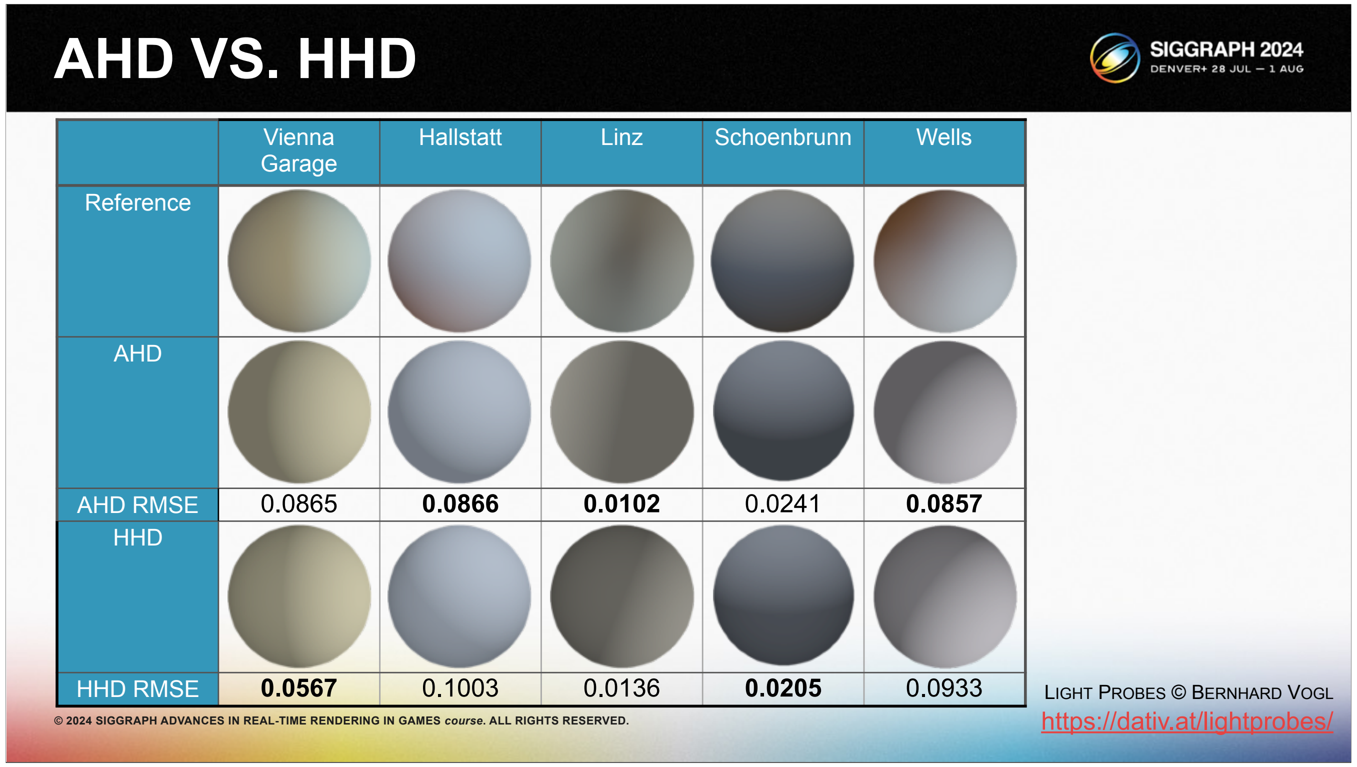
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

Optimal linear direction and spherical error
RGB RMSE: 3.795, 4.874, 5.159

Searched-for direction and hemispherical error
RGB RMSE: 1.514, 1.805, 1.597

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

You can see the difference side by side. In this case, using a searched-for direction and the hemispherical Gram matrix more than halves the reconstruction error.



AHD VS. HHD

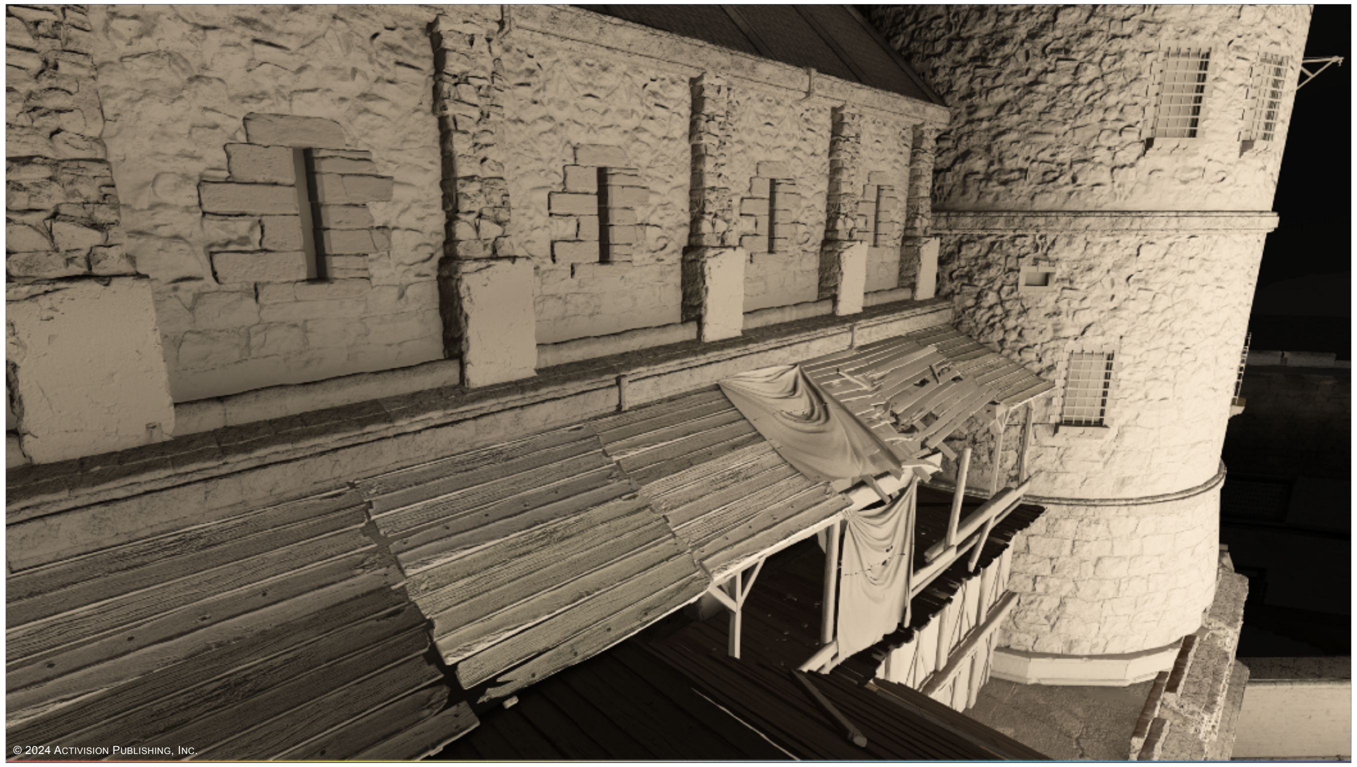
SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

	Vienna Garage	Hallstatt	Linz	Schoenbrunn	Wells
Reference					
AHD					
AHD RMSE	0.0865	**0.0866**	**0.0102**	0.0241	**0.0857**
HHD					
HHD RMSE	**0.0567**	0.1003	0.0136	**0.0205**	0.0933

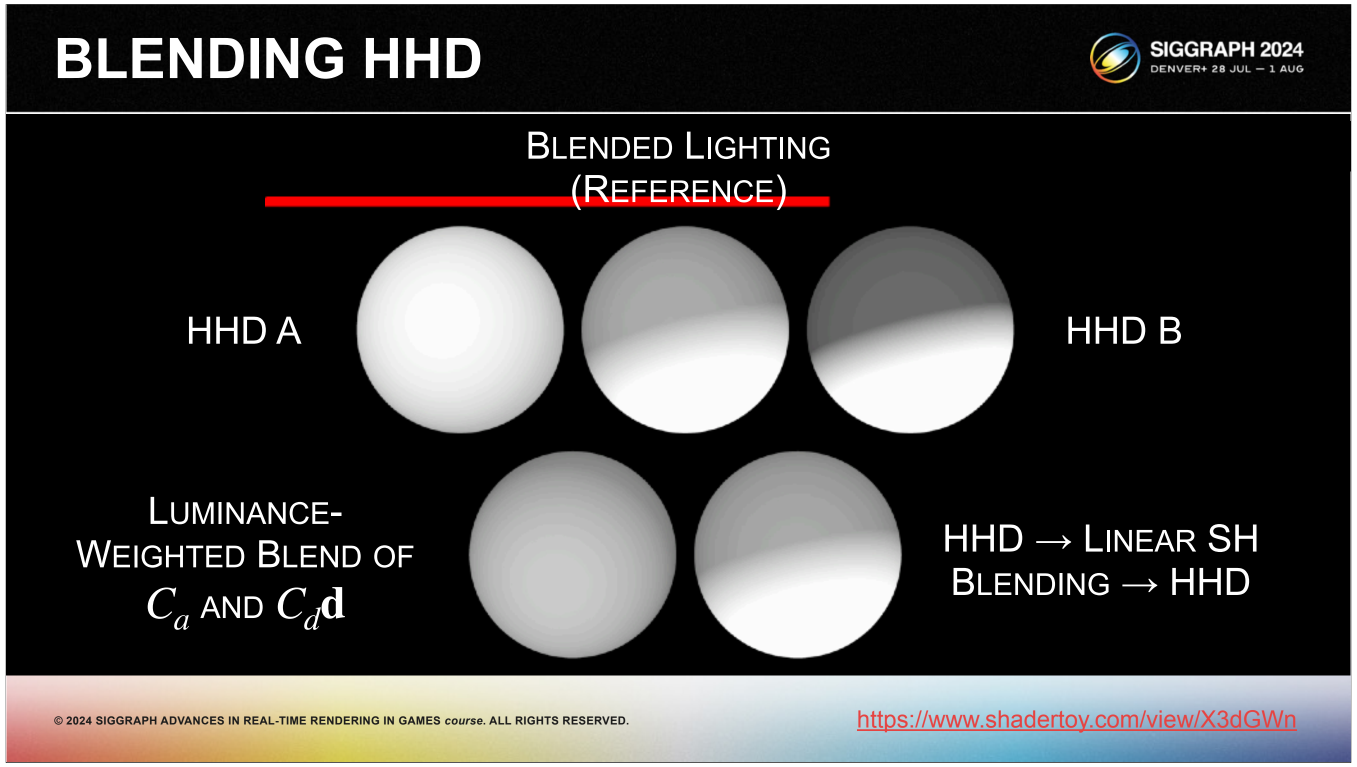
© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Light Probes © Bernhard Vogl
https://dativ.at/lightprobes/

This solve strategy works for both AHD and HHD, and it turns out that, depending on the environments, the quality difference between AHD and HHD is really pretty minor, and you can construct scenarios where either wins. Given that, it’s worth asking what makes the switch to HHD worthwhile.


© 2024 Activision Publishing, Inc.

The answer is that HHD has a big advantage over AHD when it comes to blending. In Call of Duty, our lighting isn't entirely static. We support multiple light sets that can be dynamically blended between at runtime; we might have separate lighting data for a light being on or off, or a door being open or closed. Crucially, these dynamic lighting updates are decoupled from the runtime lighting; multiple lightmaps are blended together into a single lightmap that's sampled at runtime. If you're curious about the details of that system, my colleague Peter-Pike Sloan gave a talk at Advances in 2020 which is well worth a look.



BLENDING HHD

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

Blended Lighting (Reference)

HHD A

HHD B

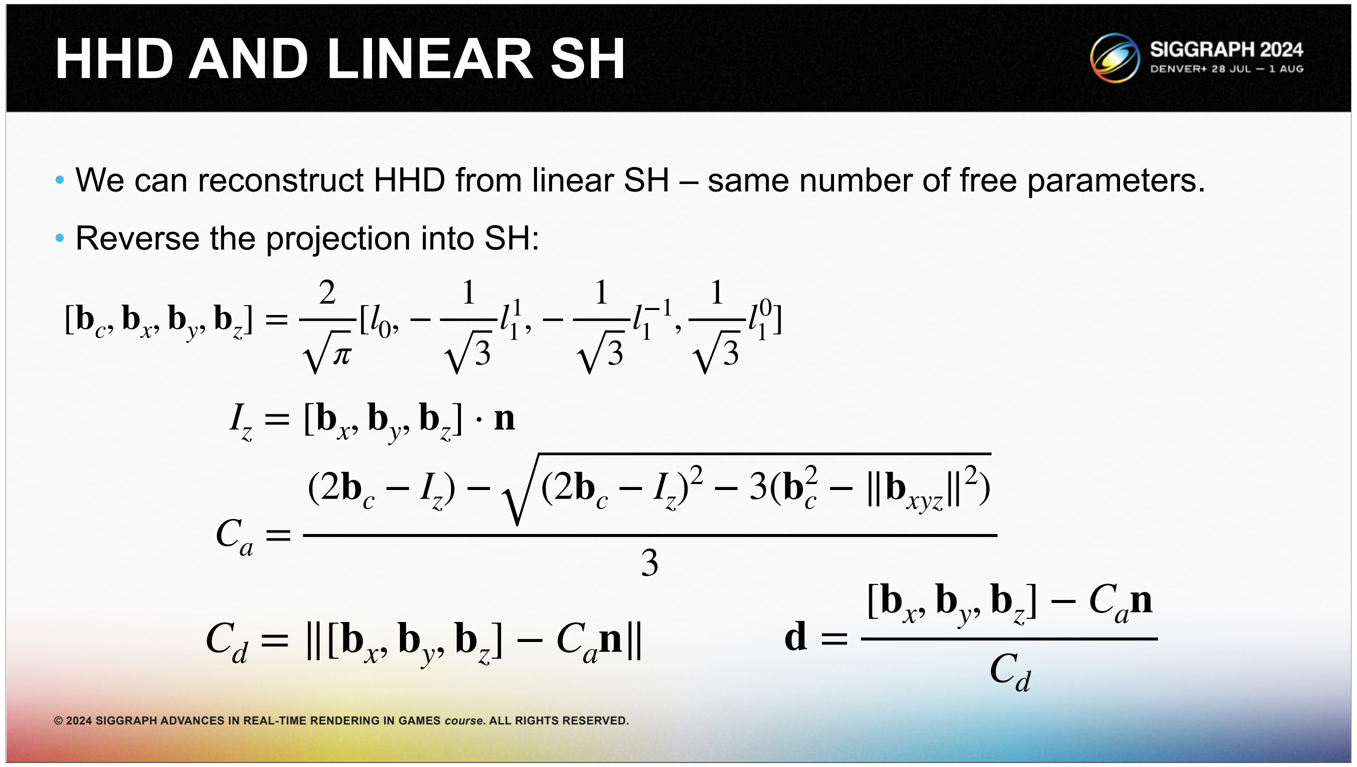
Luminance-Weighted Blend of C_a and $C_d\mathbf{d}$

HHD → Linear SH Blending → HHD

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

https://www.shadertoy.com/view/X3dGWn

Now, HHD is still a nonlinear model, and, like we saw earlier with AHD, reconstructing from the blended *parameters*, bottom left, produces results pretty far from our reference blended lighting, top centre. On the other hand, if we project to a linear model like linear SH and then solve back to HHD the results are a much closer match.



HHD AND LINEAR SH

- We can reconstruct HHD from linear SH – same number of free parameters.
- Reverse the projection into SH:

$$[\mathbf{b}_c, \mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z] = \frac{2}{\sqrt{\pi}}[l_0, -\frac{1}{\sqrt{3}}l_1^1, -\frac{1}{\sqrt{3}}l_1^{-1}, \frac{1}{\sqrt{3}}l_1^0]$$

$$I_z = [\mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z] \cdot \mathbf{n}$$

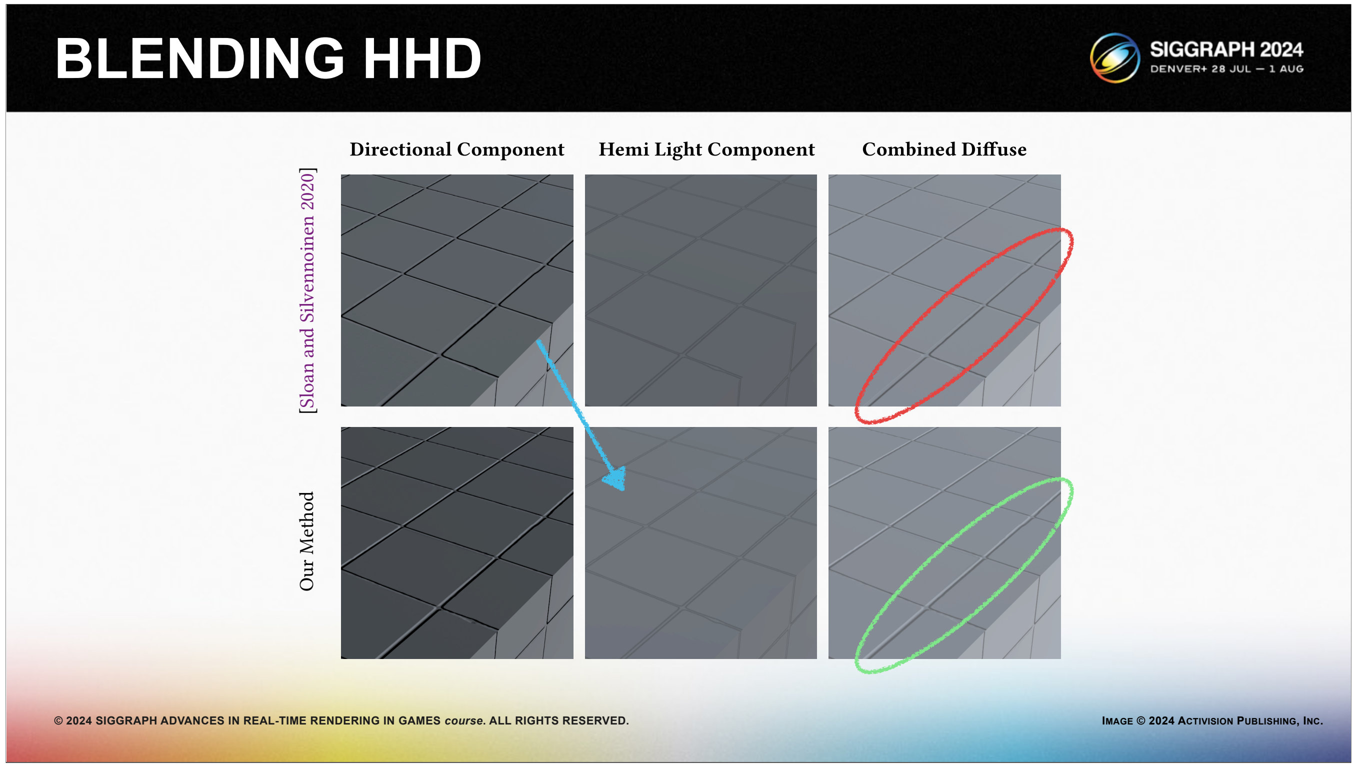
$$C_a = \frac{(2\mathbf{b}_c - I_z) - \sqrt{(2\mathbf{b}_c - I_z)^2 - 3(\mathbf{b}_c^2 - \|\mathbf{b}_{xyz}\|^2)}}{3}$$

$$C_d = \|[\mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z] - C_a\mathbf{n}\| \qquad \mathbf{d} = \frac{[\mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z] - C_a\mathbf{n}}{C_d}$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Doing this at runtime for AHD would be too expensive, but because HHD is a valid hemispherical lighting setup, it turns out that, if we have hemispherical data, we can directly and cheaply solve from linear SH to HHD.

In this case, we can get I_z directly from the linear band. Then, we simply solve a quadratic for the ambient colour and then subtract it out from the directional colour and highlight direction. This keeps our blending shaders lightweight and significantly improves the blending quality.



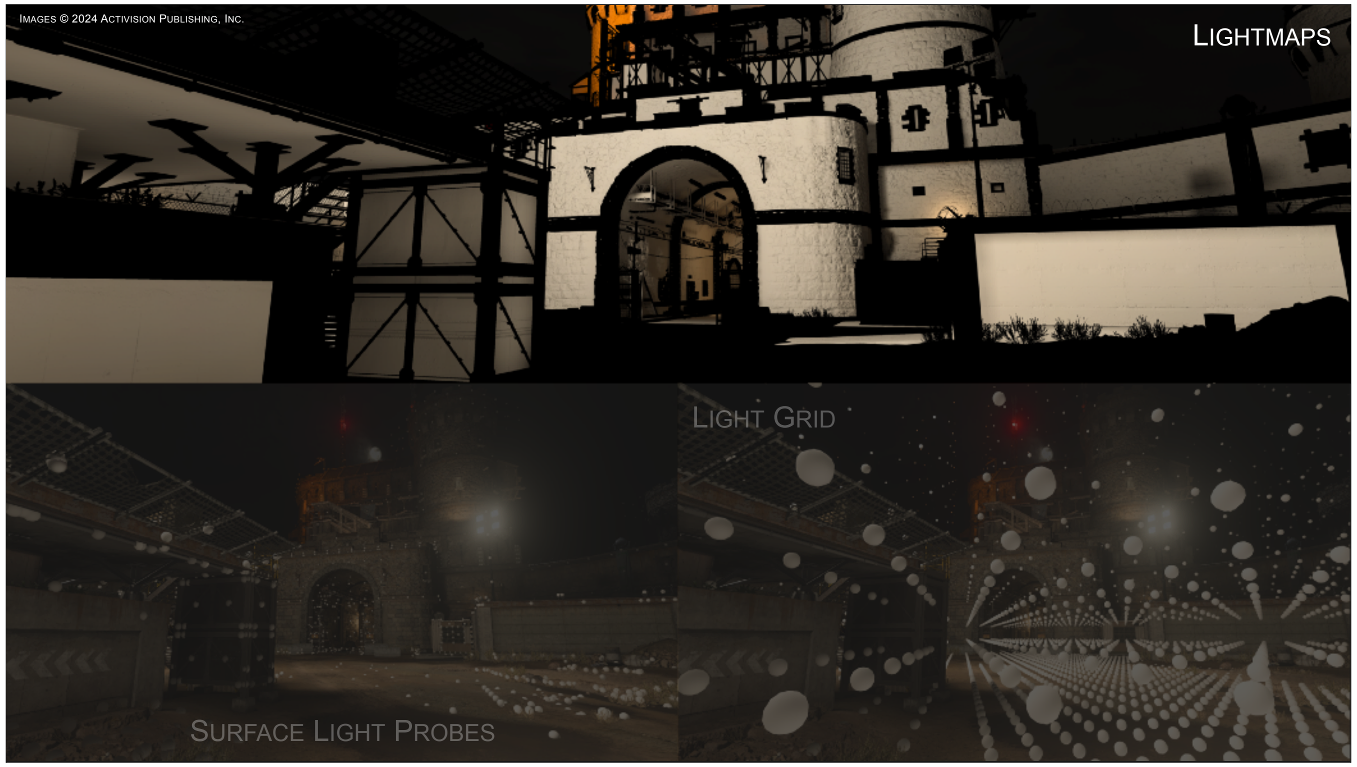
BLENDING HHD

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

IMAGE © 2024 ACTIVISION PUBLISHING, INC.

The difference is particularly obvious on steep normals. The solve moves energy from diverging directional lights into the hemisphere light so that the highlight direction can stay more grazing.



That wraps up the lightmaps portion of the talk. So far, however, we've only really discussed half of our baked lighting. Anything that's dynamic or doesn't have a nice lightmap parameterisation gets its lighting from light probes rather than lightmaps, where the lighting at each sample is represented using spherical harmonics.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

SPHERICAL HARMONICS AND OCCLUSION

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

We know how to reconstruct irradiance from spherical harmonics; just scale the bands by the cosine convolution coefficients and then reconstruct.

When normal maps are involved, though, things get a bit more complicated.

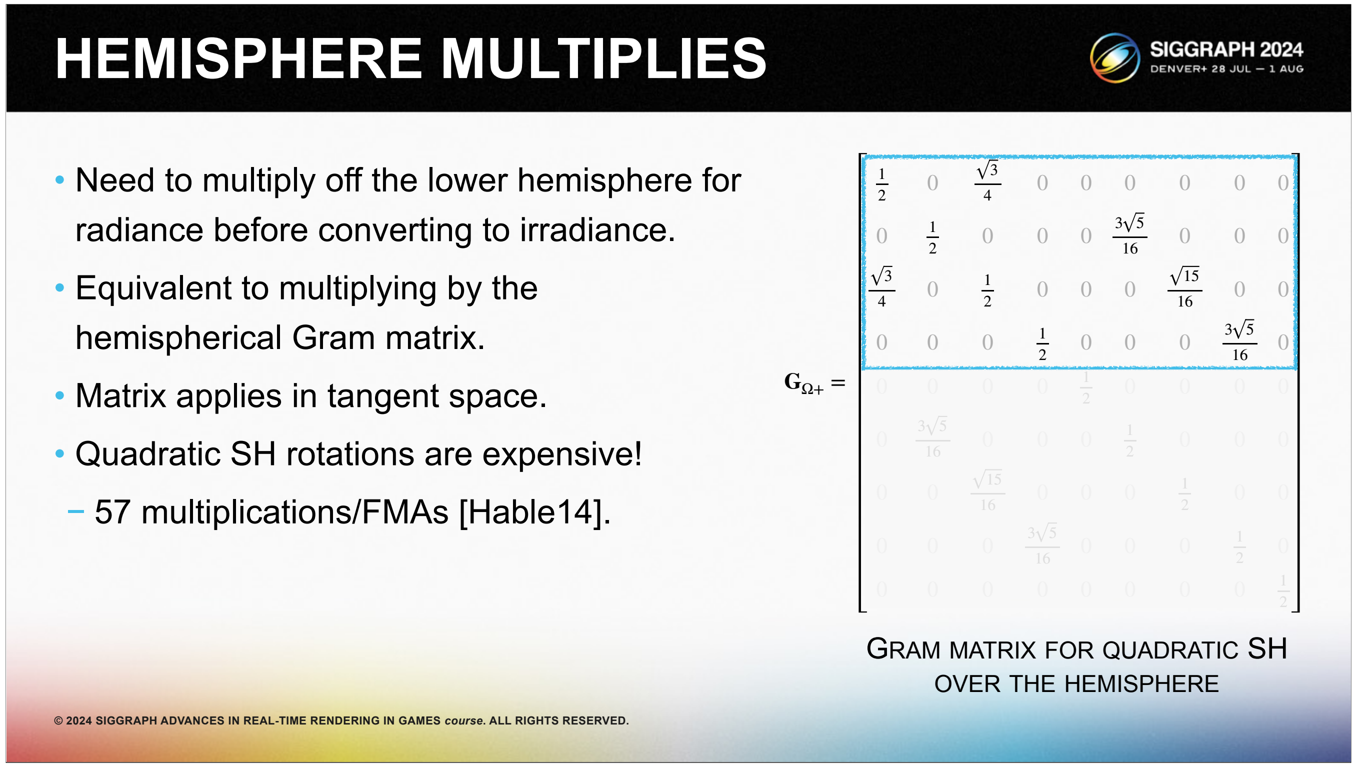

© 2024 Activision Publishing, Inc.

No Occlusion

The problem with just reconstructing irradiance in the normal direction is the same one we had with AHD: we're not accounting for hemisphere occlusion. That manifests as light leaks, and can produce an obvious visual mismatch between properly occluded lightmapped surfaces and surfaces using our SH probes, like this one.

© 2024 Activision Publishing, Inc.

What we want is something more like this; while not perfect, most of the objectionable light leakage on the normal maps is gone.



HEMISPHERE MULTIPLIES

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Need to multiply off the lower hemisphere for radiance before converting to irradiance.
- Equivalent to multiplying by the hemispherical Gram matrix.
- Matrix applies in tangent space.
- Quadratic SH rotations are expensive!
  - 57 multiplications/FMAs [Hable14].

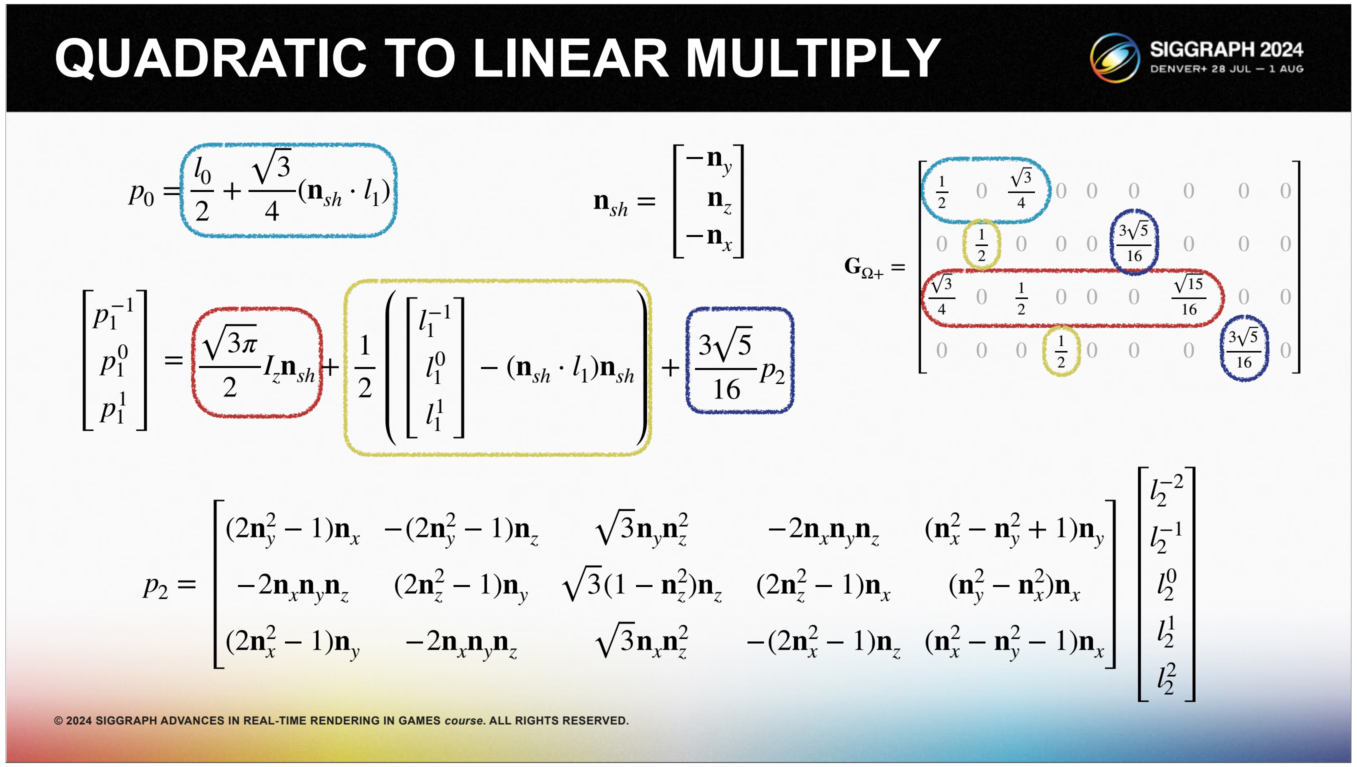
$$\mathbf{G}_{\Omega+} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{\sqrt{3}}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 \\ \frac{\sqrt{3}}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{\sqrt{15}}{16} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{15}}{16} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

GRAM MATRIX FOR QUADRATIC SH
OVER THE HEMISPHERE

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

To apply hemispherical occlusion, we need to multiply the radiance by the surface hemisphere before evaluating irradiance for normal mapped normals. One way to represent this is through multiplication with the hemispherical Gram matrix in tangent space. Unfortunately, our lighting data is in world space, and rotating a quadratic SH is a fairly heavy operation.

What if we could get away with only linear SH output from our quadratic SH input? This reduces the number of coefficients, and importantly allows us to avoid the rotations altogether. To see how this works, let’s decompose our tangent-space hemisphere multiplication matrix.



QUADRATIC TO LINEAR MULTIPLY

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

$$p_0 = \boxed{\frac{l_0}{2} + \frac{\sqrt{3}}{4}(\mathbf{n}_{sh} \cdot l_1)} \qquad \mathbf{n}_{sh} = \begin{bmatrix} -\mathbf{n}_y \\ \mathbf{n}_z \\ -\mathbf{n}_x \end{bmatrix}$$

$$\mathbf{G}_{\Omega+} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{\sqrt{3}}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 & 0 & 0 \\ \frac{\sqrt{3}}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{\sqrt{15}}{16} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{3\sqrt{5}}{16} & 0 \end{bmatrix}$$

$$\begin{bmatrix} p_1^{-1} \\ p_1^0 \\ p_1^1 \end{bmatrix} = \boxed{\frac{\sqrt{3\pi}}{2} I_z \mathbf{n}_{sh}} + \boxed{\frac{1}{2}\left(\begin{bmatrix} l_1^{-1} \\ l_1^0 \\ l_1^1 \end{bmatrix} - (\mathbf{n}_{sh} \cdot l_1)\mathbf{n}_{sh}\right)} + \boxed{\frac{3\sqrt{5}}{16} p_2}$$

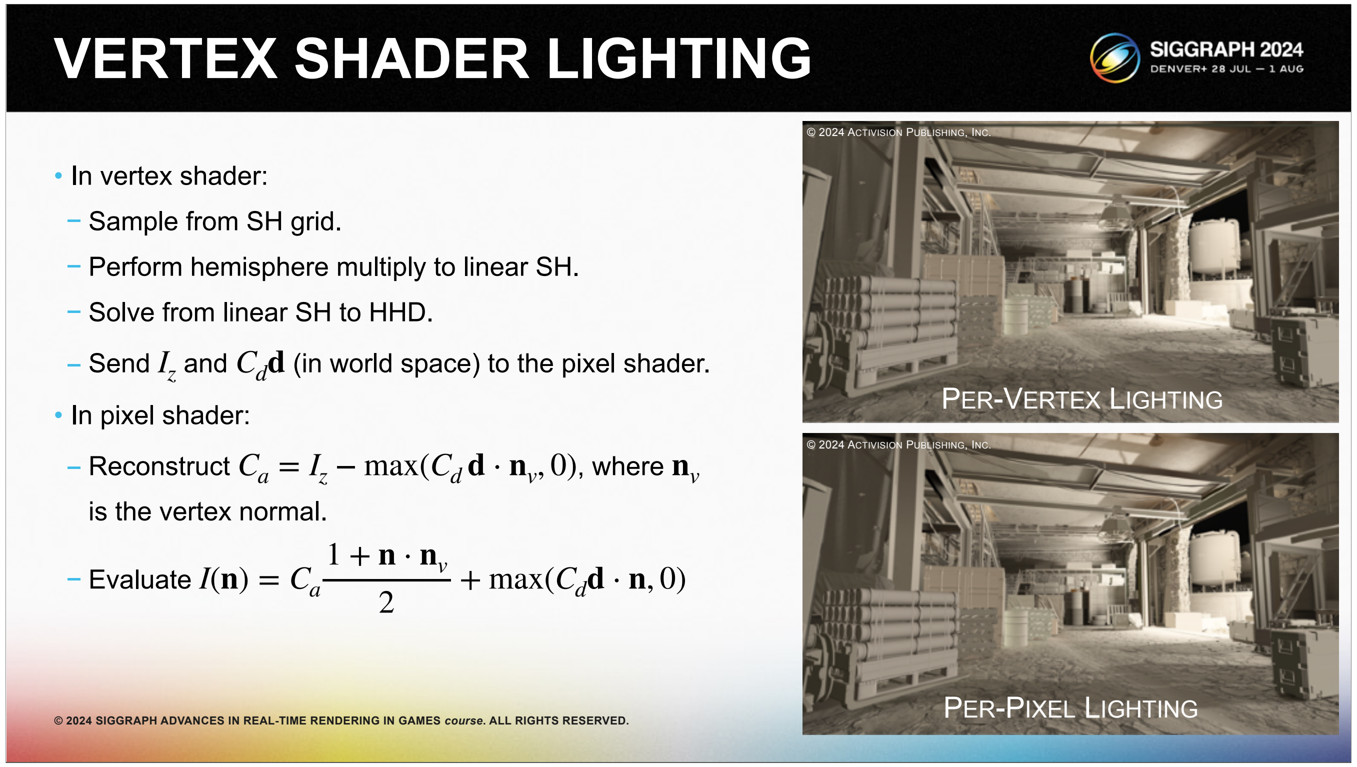
$$p_2 = \begin{bmatrix} (2\mathbf{n}_y^2 - 1)\mathbf{n}_x & -(2\mathbf{n}_y^2 - 1)\mathbf{n}_z & \sqrt{3}\mathbf{n}_y\mathbf{n}_z^2 & -2\mathbf{n}_x\mathbf{n}_y\mathbf{n}_z & (\mathbf{n}_x^2 - \mathbf{n}_y^2 + 1)\mathbf{n}_y \\ -2\mathbf{n}_x\mathbf{n}_y\mathbf{n}_z & (2\mathbf{n}_z^2 - 1)\mathbf{n}_y & \sqrt{3}(1 - \mathbf{n}_z^2)\mathbf{n}_z & (2\mathbf{n}_z^2 - 1)\mathbf{n}_x & (\mathbf{n}_y^2 - \mathbf{n}_x^2)\mathbf{n}_x \\ (2\mathbf{n}_x^2 - 1)\mathbf{n}_y & -2\mathbf{n}_x\mathbf{n}_y\mathbf{n}_z & \sqrt{3}\mathbf{n}_x\mathbf{n}_z^2 & -(2\mathbf{n}_x^2 - 1)\mathbf{n}_z & (\mathbf{n}_x^2 - \mathbf{n}_y^2 - 1)\mathbf{n}_x \end{bmatrix} \begin{bmatrix} l_2^{-2} \\ l_2^{-1} \\ l_2^0 \\ l_2^1 \\ l_2^2 \end{bmatrix}$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

We can compute the constant term from the input L0 band and the tangent-space zonal L1 coefficient, which is just the dot product of the linear band with a swizzled normal vector.

The linear band is a bit more complex. The tangent-space zonal component is just the scaled irradiance in the hemisphere normal. Then, we need to scale the non-zonal components of the linear band. Finally, we add the projected components of the quadratic band.

It turns out to be possible to extract these components in an oriented frame. This gives us an efficient quadratic to linear hemisphere multiply.



VERTEX SHADER LIGHTING

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- In vertex shader:
  - Sample from SH grid.
  - Perform hemisphere multiply to linear SH.
  - Solve from linear SH to HHD.
  - Send I_z and $C_d\mathbf{d}$ (in world space) to the pixel shader.
- In pixel shader:
  - Reconstruct $C_a = I_z - \max(C_d\,\mathbf{d}\cdot\mathbf{n}_v, 0)$, where $\mathbf{n}_v$ is the vertex normal.
  - Evaluate $I(\mathbf{n}) = C_a\dfrac{1+\mathbf{n}\cdot\mathbf{n}_v}{2} + \max(C_d\mathbf{d}\cdot\mathbf{n}, 0)$

© 2024 Activision Publishing, Inc.

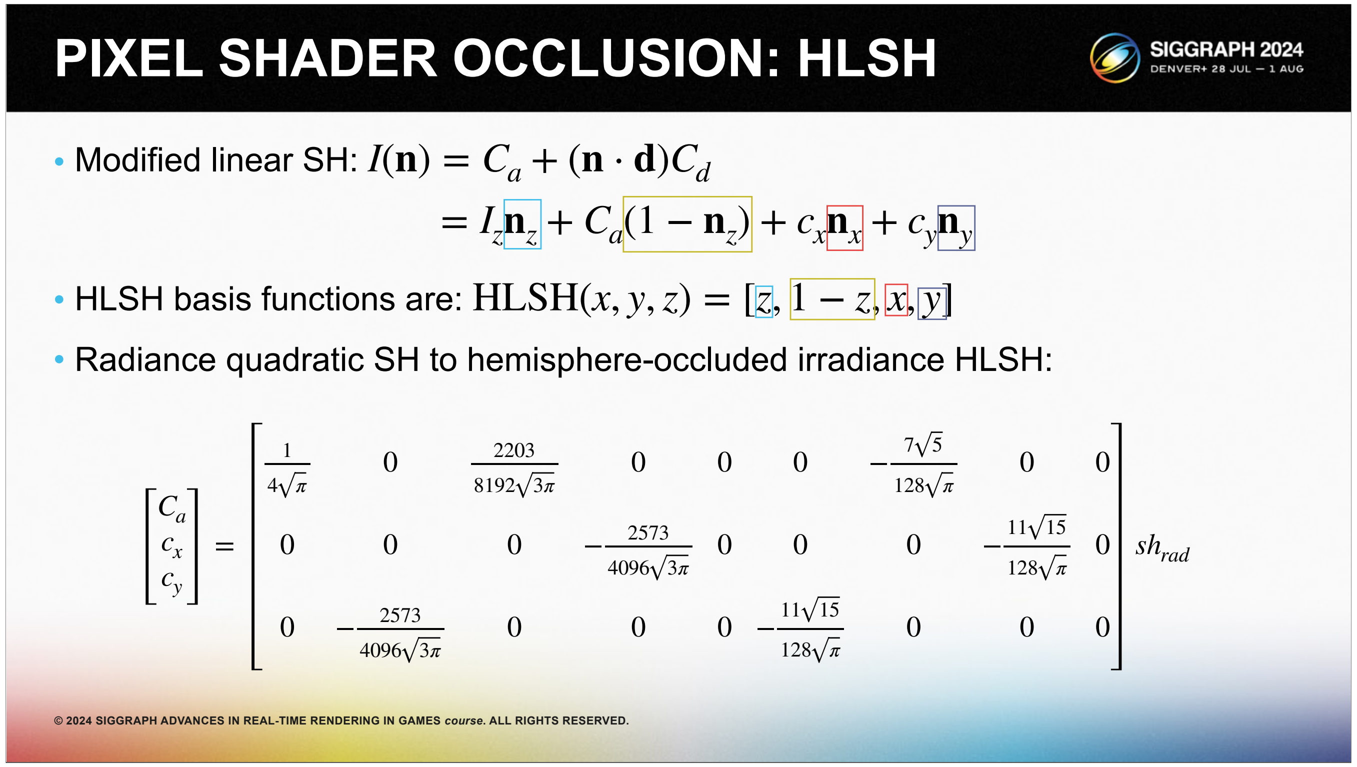
Per-Vertex Lighting

© 2024 Activision Publishing, Inc.

Per-Pixel Lighting

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

On low-end platforms, we perform this in the vertex shader, sampling from SH, multiplying by a hemisphere, and then solving to luminance HHD which gets passed to the pixel shader; this helps to reduce our lighting costs while still retaining normal map variation.



PIXEL SHADER OCCLUSION: HLSH

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Modified linear SH: $I(\mathbf{n}) = C_a + (\mathbf{n} \cdot \mathbf{d})C_d$

$$= I_z\boxed{\mathbf{n}_z} + C_a\boxed{(1 - \mathbf{n}_z)} + c_x\boxed{\mathbf{n}_x} + c_y\boxed{\mathbf{n}_y}$$

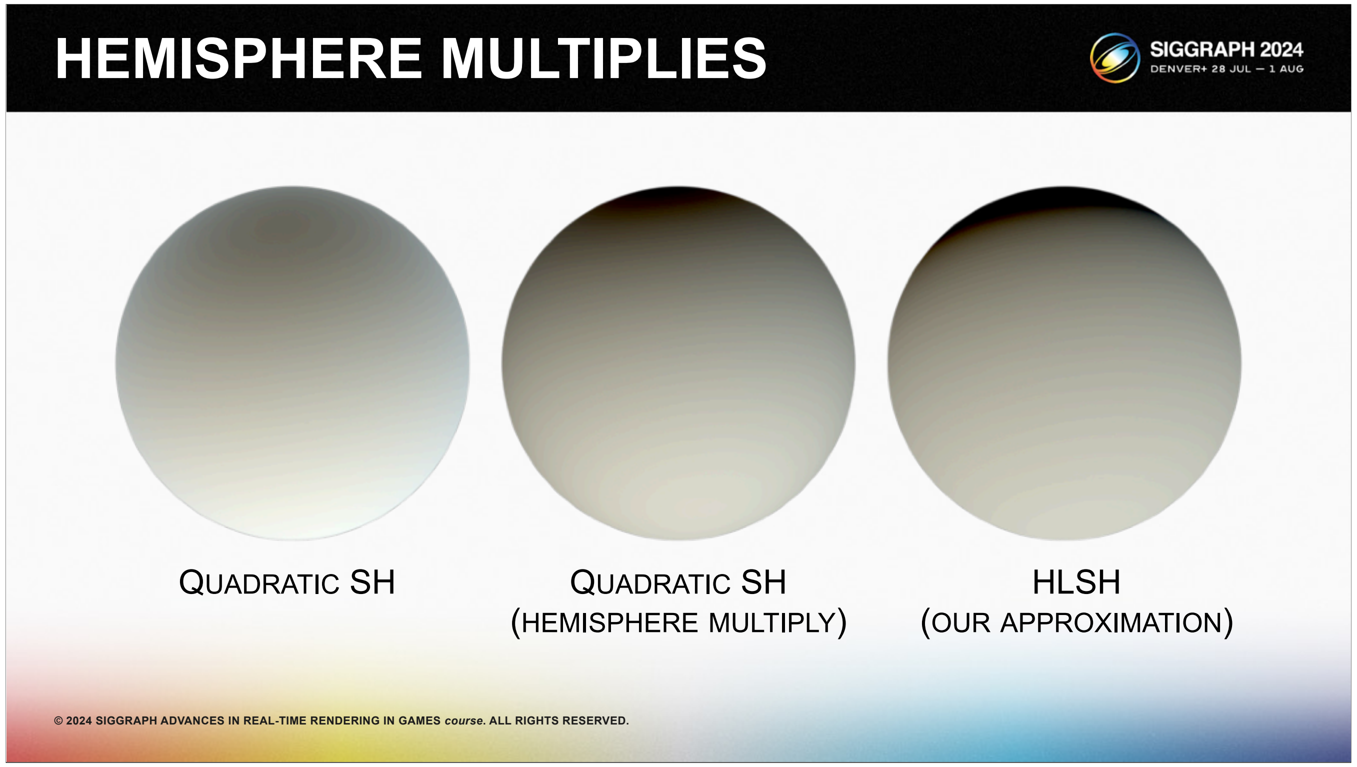
- HLSH basis functions are: $\mathrm{HLSH}(x, y, z) = [\boxed{z}, \boxed{1 - z}, \boxed{x}, \boxed{y}]$
- Radiance quadratic SH to hemisphere-occluded irradiance HLSH:

$$\begin{bmatrix} C_a \\ c_x \\ c_y \end{bmatrix} = \begin{bmatrix} \frac{1}{4\sqrt{\pi}} & 0 & \frac{2203}{8192\sqrt{3\pi}} & 0 & 0 & 0 & -\frac{7\sqrt{5}}{128\sqrt{\pi}} & 0 & 0 \\ 0 & 0 & 0 & -\frac{2573}{4096\sqrt{3\pi}} & 0 & 0 & 0 & -\frac{11\sqrt{15}}{128\sqrt{\pi}} & 0 \\ 0 & -\frac{2573}{4096\sqrt{3\pi}} & 0 & 0 & 0 & -\frac{11\sqrt{15}}{128\sqrt{\pi}} & 0 & 0 & 0 \end{bmatrix} sh_{rad}$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

On higher end, we perform the multiply directly in the pixel shader. To extract a bit more quality, we actually solve to a scaled version of linear SH over the hemisphere with the IrradZ constraint that we call HLSH, where the weight for the z basis function is always given by IrradZ.

The matrix multiplication to produce HLSH uses the same quadratic SH components as our optimised hemisphere multiply from before, so we can use the same technique to perform the projection in world space.



HEMISPHERE MULTIPLIES

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

QUADRATIC SH

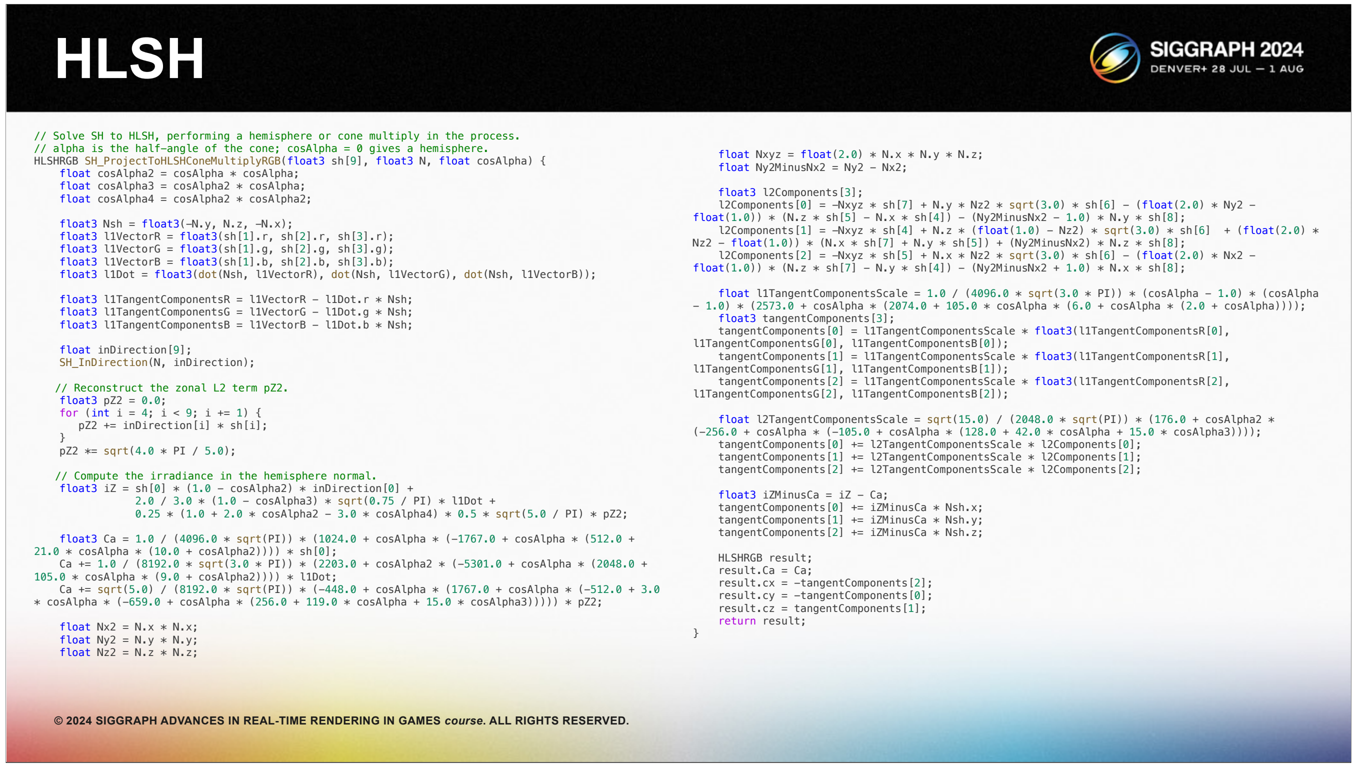
QUADRATIC SH
(HEMISPHERE MULTIPLY)

HLSH
(OUR APPROXIMATION)

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Here’s what that looks like. We have quadratic SH with no occlusion on the left, a full hemisphere multiply in the centre, and HLSH on the right, all showing irradiance over the hemisphere. We keep the IrradZ constraint, so the hemisphere normal is exact between the full hemisphere multiply and our approximation.

Our approximation loses some of the skew that full quadratic SH can have, but it’s a pretty close match otherwise, and doesn’t suffer as badly from ringing.



HLSH

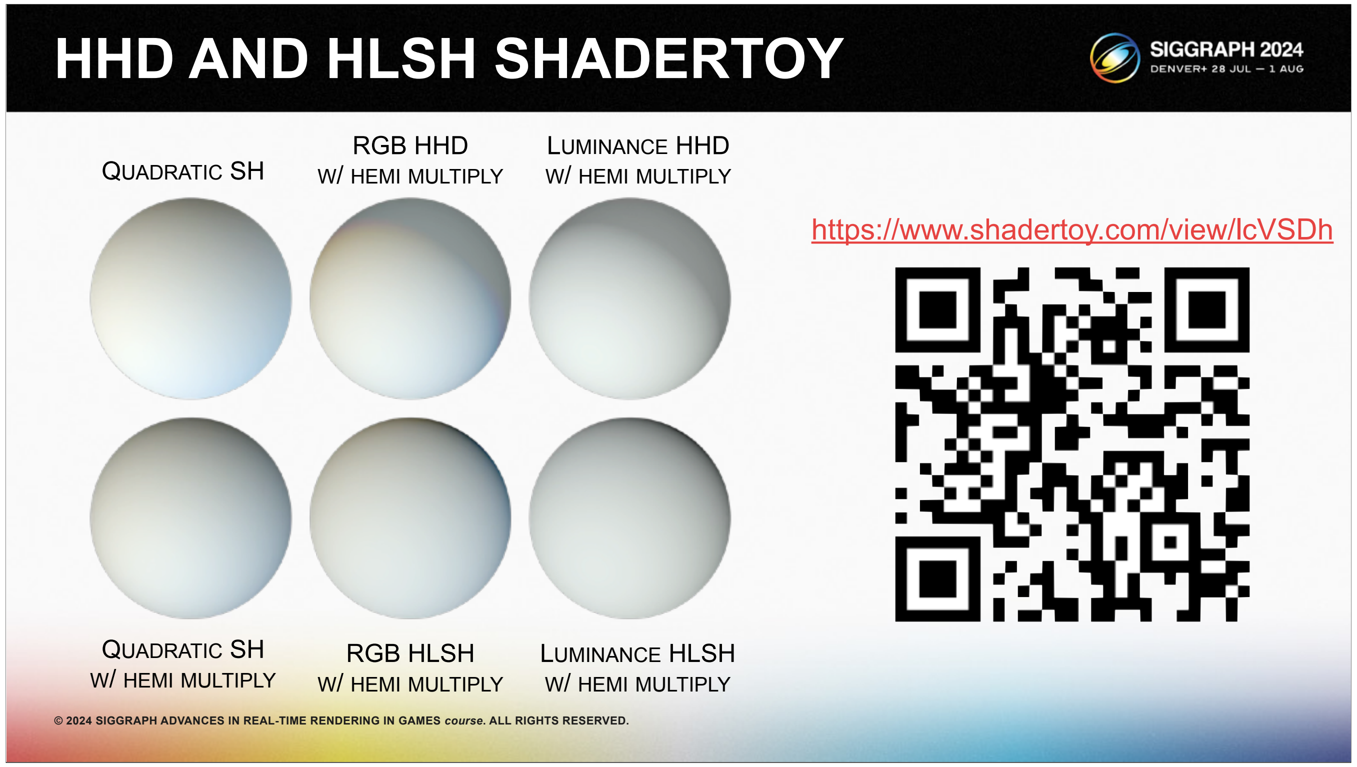
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

```
// Solve SH to HLSH, performing a hemisphere or cone multiply in the process.
// alpha is the half-angle of the cone; cosAlpha = 0 gives a hemisphere.
HLSHRGB SH_ProjectToHLSHConeMultiplyRGB(float3 sh[9], float3 N, float cosAlpha) {
    float cosAlpha2 = cosAlpha * cosAlpha;
    float cosAlpha3 = cosAlpha2 * cosAlpha;
    float cosAlpha4 = cosAlpha2 * cosAlpha2;

    float3 Nsh = float3(-N.y, N.z, -N.x);
    float3 l1VectorR = float3(sh[1].r, sh[2].r, sh[3].r);
    float3 l1VectorG = float3(sh[1].g, sh[2].g, sh[3].g);
    float3 l1VectorB = float3(sh[1].b, sh[2].b, sh[3].b);
    float3 l1Dot = float3(dot(Nsh, l1VectorR), dot(Nsh, l1VectorG), dot(Nsh, l1VectorB));

    float3 l1TangentComponentsR = l1VectorR - l1Dot.r * Nsh;
    float3 l1TangentComponentsG = l1VectorG - l1Dot.g * Nsh;
    float3 l1TangentComponentsB = l1VectorB - l1Dot.b * Nsh;

    float inDirection[9];
    SH_InDirection(N, inDirection);

   // Reconstruct the zonal L2 term pZ2.
    float3 pZ2 = 0.0;
    for (int i = 4; i < 9; i += 1) {
       pZ2 += inDirection[i] * sh[i];
    }
    pZ2 *= sqrt(4.0 * PI / 5.0);

   // Compute the irradiance in the hemisphere normal.
    float3 iZ = sh[0] * (1.0 - cosAlpha2) * inDirection[0] +
                2.0 / 3.0 * (1.0 - cosAlpha3) * sqrt(0.75 / PI) * l1Dot +
                0.25 * (1.0 + 2.0 * cosAlpha2 - 3.0 * cosAlpha4) * 0.5 * sqrt(5.0 / PI) * pZ2;

    float3 Ca = 1.0 / (4096.0 * sqrt(PI)) * (1024.0 + cosAlpha * (-1767.0 + cosAlpha * (512.0 +
21.0 * cosAlpha * (10.0 + cosAlpha2)))) * sh[0];
    Ca += 1.0 / (8192.0 * sqrt(3.0 * PI)) * (2203.0 + cosAlpha2 * (-5301.0 + cosAlpha * (2048.0 +
105.0 * cosAlpha * (9.0 + cosAlpha2)))) * l1Dot;
    Ca += sqrt(5.0) / (8192.0 * sqrt(PI)) * (-448.0 + cosAlpha * (1767.0 + cosAlpha * (-512.0 + 3.0
* cosAlpha * (-659.0 + cosAlpha * (256.0 + 119.0 * cosAlpha + 15.0 * cosAlpha3))))) * pZ2;

    float Nx2 = N.x * N.x;
    float Ny2 = N.y * N.y;
    float Nz2 = N.z * N.z;

    float Nxyz = float(2.0) * N.x * N.y * N.z;
    float Ny2MinusNx2 = Ny2 - Nx2;

    float3 l2Components[3];
    l2Components[0] = -Nxyz * sh[7] + N.y * Nz2 * sqrt(3.0) * sh[6] - (float(2.0) * Ny2 -
float(1.0)) * (N.z * sh[5] - N.x * sh[4]) - (Ny2MinusNx2 - 1.0) * N.y * sh[8];
    l2Components[1] = -Nxyz * sh[4] + N.z * (float(1.0) - Nz2) * sqrt(3.0) * sh[6]  + (float(2.0) *
Nz2 - float(1.0)) * (N.x * sh[7] + N.y * sh[5]) + (Ny2MinusNx2) * N.z * sh[8];
    l2Components[2] = -Nxyz * sh[5] + N.x * Nz2 * sqrt(3.0) * sh[6] - (float(2.0) * Nx2 -
float(1.0)) * (N.z * sh[7] - N.y * sh[4]) - (Ny2MinusNx2 + 1.0) * N.x * sh[8];

    float l1TangentComponentsScale = 1.0 / (4096.0 * sqrt(3.0 * PI)) * (cosAlpha - 1.0) * (cosAlpha
- 1.0) * (2573.0 + cosAlpha * (2074.0 + 105.0 * cosAlpha * (6.0 + cosAlpha * (2.0 + cosAlpha))));
    float3 tangentComponents[3];
    tangentComponents[0] = l1TangentComponentsScale * float3(l1TangentComponentsR[0],
l1TangentComponentsG[0], l1TangentComponentsB[0]);
    tangentComponents[1] = l1TangentComponentsScale * float3(l1TangentComponentsR[1],
l1TangentComponentsG[1], l1TangentComponentsB[1]);
    tangentComponents[2] = l1TangentComponentsScale * float3(l1TangentComponentsR[2],
l1TangentComponentsG[2], l1TangentComponentsB[2]);

    float l2TangentComponentsScale = sqrt(15.0) / (2048.0 * sqrt(PI)) * (176.0 + cosAlpha2 *
(-256.0 + cosAlpha * (-105.0 + cosAlpha * (128.0 + 42.0 * cosAlpha + 15.0 * cosAlpha3))));
    tangentComponents[0] += l2TangentComponentsScale * l2Components[0];
    tangentComponents[1] += l2TangentComponentsScale * l2Components[1];
    tangentComponents[2] += l2TangentComponentsScale * l2Components[2];

    float3 iZMinusCa = iZ - Ca;
    tangentComponents[0] += iZMinusCa * Nsh.x;
    tangentComponents[1] += iZMinusCa * Nsh.y;
    tangentComponents[2] += iZMinusCa * Nsh.z;

    HLSHRGB result;
    result.Ca = Ca;
    result.cx = -tangentComponents[2];
    result.cy = -tangentComponents[0];
    result.cz = tangentComponents[1];
    return result;
}
```

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

This is what the code looks like; a lot of constant factors, so it all folds down fairly efficiently.

If you have keen eyes you may notice an extra cosAlpha parameter here; we'll get to that in a moment, but it's set to zero for hemispheres.



HHD AND HLSH SHADERTOY

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

QUADRATIC SH

RGB HHD
W/ HEMI MULTIPLY

LUMINANCE HHD
W/ HEMI MULTIPLY

QUADRATIC SH
W/ HEMI MULTIPLY

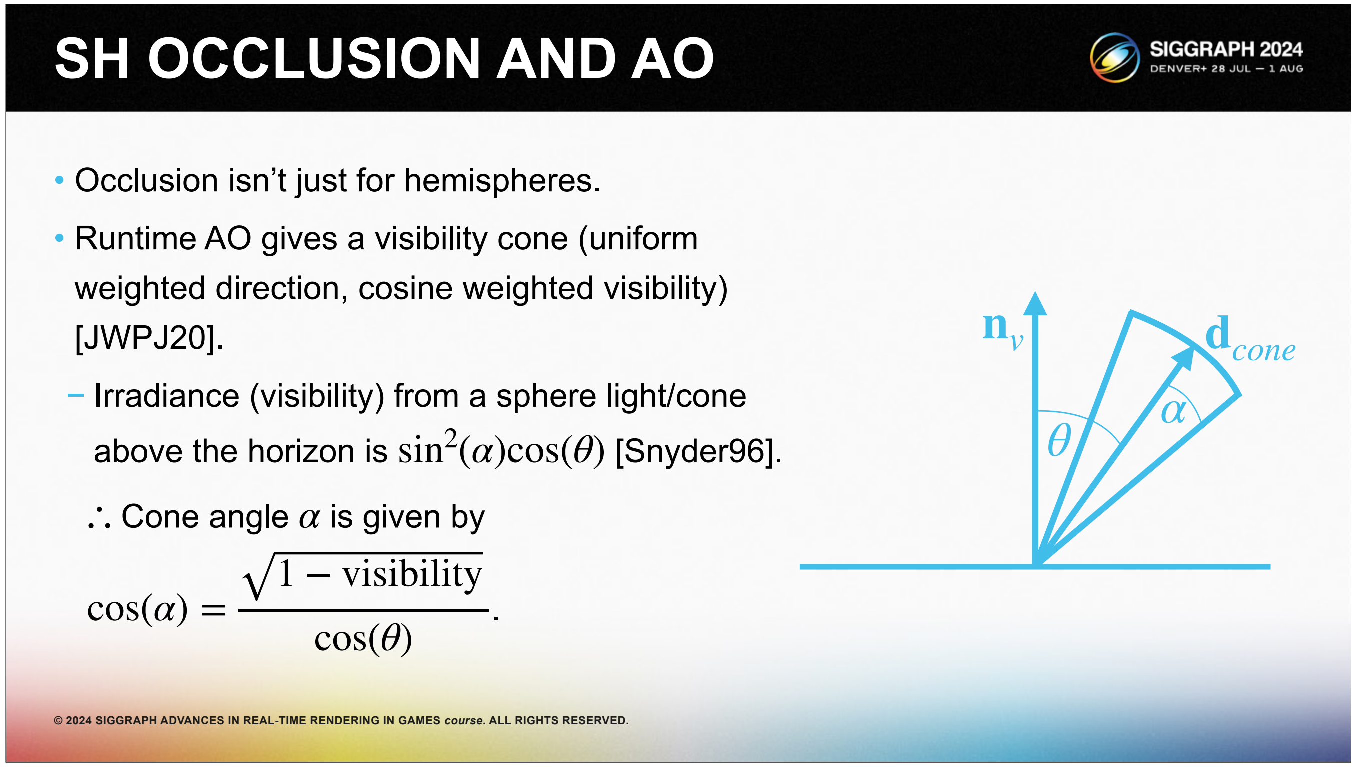
RGB HLSH
W/ HEMI MULTIPLY

LUMINANCE HLSH
W/ HEMI MULTIPLY

https://www.shadertoy.com/view/lcVSDh

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

That code is taken from a ShaderToy, which also has implementations for the oriented quadratic to linear hemisphere multiply and the HHD solve from linear spherical harmonics.



SH OCCLUSION AND AO

- Occlusion isn't just for hemispheres.
- Runtime AO gives a visibility cone (uniform weighted direction, cosine weighted visibility) [JWPJ20].
  - Irradiance (visibility) from a sphere light/cone above the horizon is $\sin^2(\alpha)\cos(\theta)$ [Snyder96].

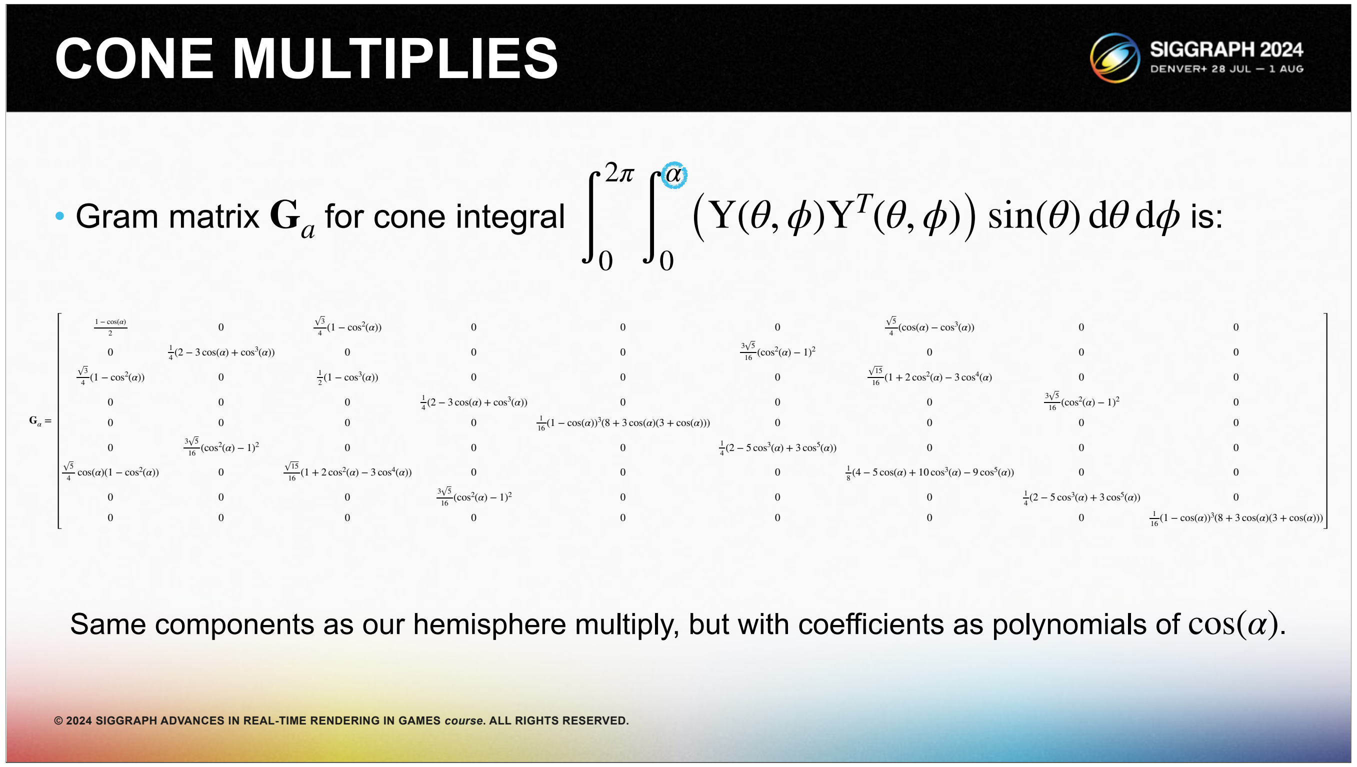
$\therefore$ Cone angle α is given by

$$\cos(\alpha) = \frac{\sqrt{1 - \text{visibility}}}{\cos(\theta)}.$$

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

We now have a lighting setup with correct hemispherical occlusion and optimal error across objects lit by both lightmaps and SH. Why stop there, though? We have another dynamic component for our indirect lighting: ambient occlusion, which gives us both an occlusion factor for the scalar irradiance and an average unoccluded direction. A cone of radiance is equivalent to a sphere light, and we can analytically convert the cosine-weighted visibility to the cone angle alpha.

We need to somehow integrate this visibility cone with our indirect lighting. In the past, bent normals have been commonly used, but that's a fairly coarse approximation. Instead, what if we can multiply our spherical lighting directly by the visibility cone?



CONE MULTIPLIES

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

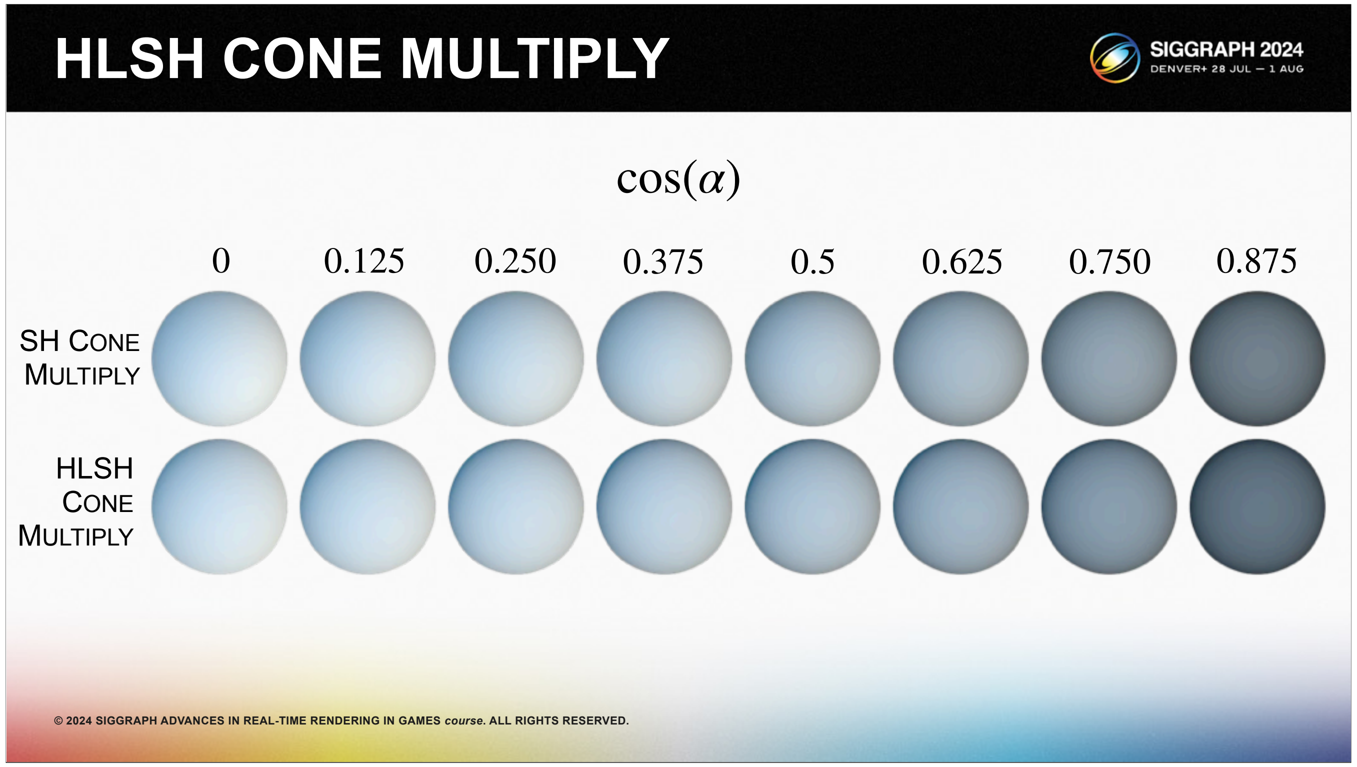
- Gram matrix $\mathbf{G}_a$ for cone integral $\int_0^{2\pi}\int_0^{\alpha}\left(\mathrm{Y}(\theta,\phi)\mathrm{Y}^T(\theta,\phi)\right)\sin(\theta)\,\mathrm{d}\theta\,\mathrm{d}\phi$ is:

$$\mathbf{G}_\alpha = \begin{bmatrix}
\frac{1-\cos(\alpha)}{2} & 0 & \frac{\sqrt{3}}{4}(1-\cos^2(\alpha)) & 0 & 0 & 0 & \frac{\sqrt{5}}{4}(\cos(\alpha)-\cos^3(\alpha)) & 0 & 0 \\
0 & \frac{1}{4}(2-3\cos(\alpha)+\cos^3(\alpha)) & 0 & 0 & 0 & \frac{3\sqrt{5}}{16}(\cos^2(\alpha)-1)^2 & 0 & 0 & 0 \\
\frac{\sqrt{3}}{4}(1-\cos^2(\alpha)) & 0 & \frac{1}{2}(1-\cos^3(\alpha)) & 0 & 0 & 0 & \frac{\sqrt{15}}{16}(1+2\cos^2(\alpha)-3\cos^4(\alpha)) & 0 & 0 \\
0 & 0 & 0 & \frac{1}{4}(2-3\cos(\alpha)+\cos^3(\alpha)) & 0 & 0 & 0 & \frac{3\sqrt{5}}{16}(\cos^2(\alpha)-1)^2 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{16}(1-\cos(\alpha))^3(8+3\cos(\alpha)(3+\cos(\alpha))) & 0 & 0 & 0 & 0 \\
0 & \frac{3\sqrt{5}}{16}(\cos^2(\alpha)-1)^2 & 0 & 0 & 0 & \frac{1}{4}(2-5\cos^3(\alpha)+3\cos^5(\alpha)) & 0 & 0 & 0 \\
\frac{\sqrt{5}}{4}\cos(\alpha)(1-\cos^2(\alpha)) & 0 & \frac{\sqrt{15}}{16}(1+2\cos^2(\alpha)-3\cos^4(\alpha)) & 0 & 0 & 0 & \frac{1}{8}(4-5\cos(\alpha)+10\cos^3(\alpha)-9\cos^5(\alpha)) & 0 & 0 \\
0 & 0 & 0 & \frac{3\sqrt{5}}{16}(\cos^2(\alpha)-1)^2 & 0 & 0 & 0 & \frac{1}{4}(2-5\cos^3(\alpha)+3\cos^5(\alpha)) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{16}(1-\cos(\alpha))^3(8+3\cos(\alpha)(3+\cos(\alpha)))
\end{bmatrix}$$

Same components as our hemisphere multiply, but with coefficients as polynomials of $\cos(\alpha)$.

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Our hemisphere multiply is given by integrating theta from 0 to pi over 2. If we assume the visibility cone is oriented with the hemisphere normal, the matrix for the cone is derived by simply integrating to the cone angle instead. By swapping in this matrix, our hemisphere multiply becomes a cone multiply.



HLSH CONE MULTIPLY

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

You can see that using this cone multiply as part of HLSH yields pretty close visual results to the full quadratic SH multiply. We're looking at cos(alpha) here, so the cone angle gets narrower as cos(alpha) increases, and 0 is the hemisphere case.

We always take the normal for the multiply to be the cone centre. If the cone centre is oriented away from the hemisphere normal, this approximation is less accurate, but HLSH is fairly well behaved so the error isn't visually objectionable. Compared to just using the scalar irradiance for AO, this can yield pretty dramatic visual improvements.

Let’s go back to our example from before. This is the column without any occlusion.

© 2024 ACTIVISION PUBLISHING, INC.

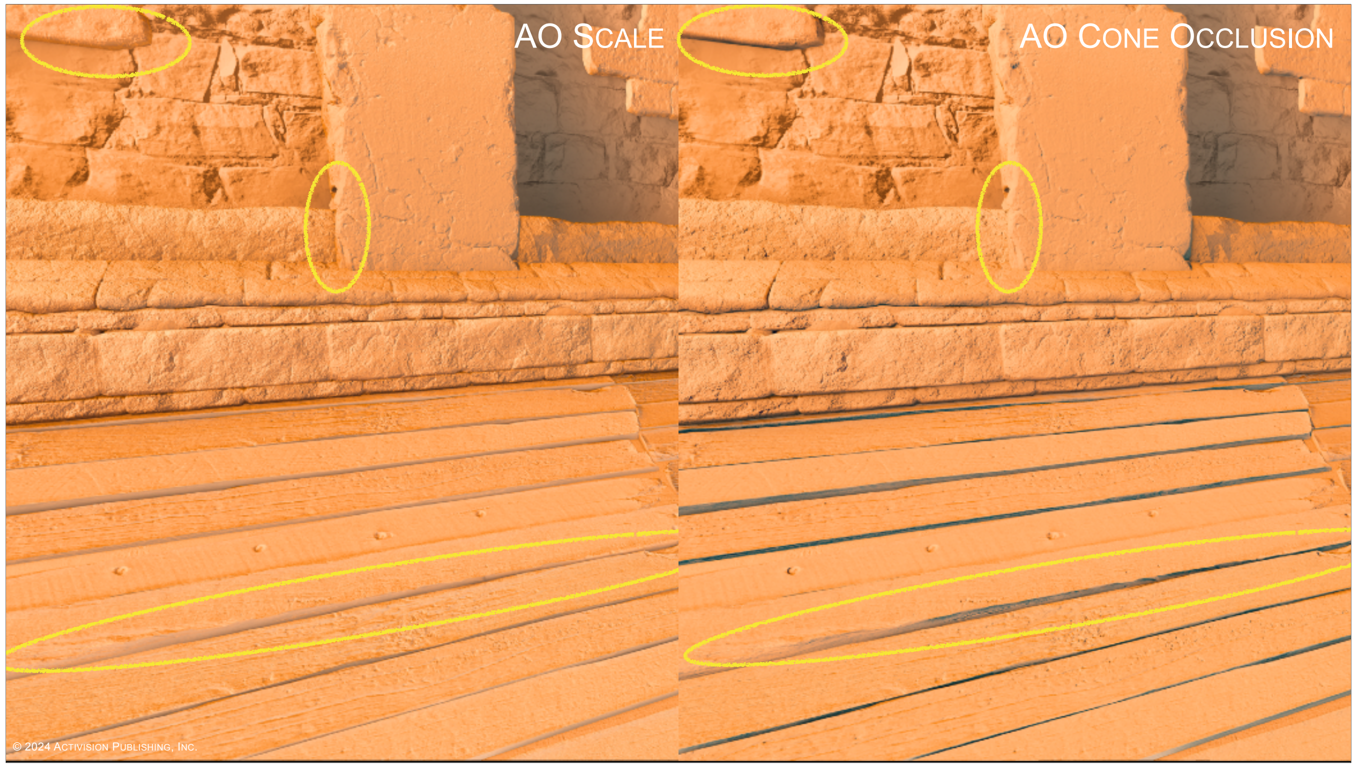
Adding hemisphere occlusion cleans up most of the light leaks, but there are still a couple of spots that have an unrealistic glow.


© 2024 Activision Publishing, Inc.

Just scaling the irradiance by the AO visibility doesn’t help here, and we’ve got our light leaking back from the normal map.

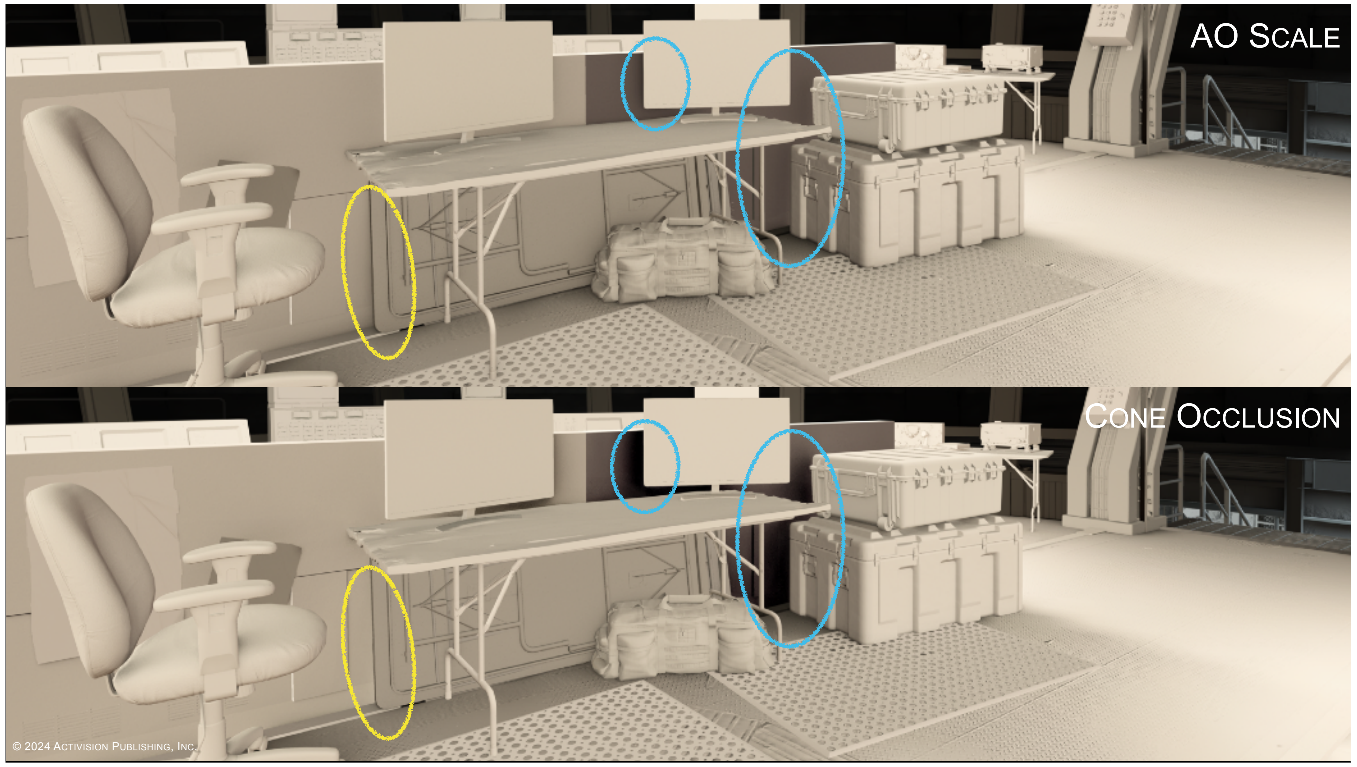
© 2024 Activision Publishing, Inc.

By contrast, using HLSH occlusion with a visibility cone yields massive improvements. The lighting is now looking properly integrated with the lightmapped wall beside it.



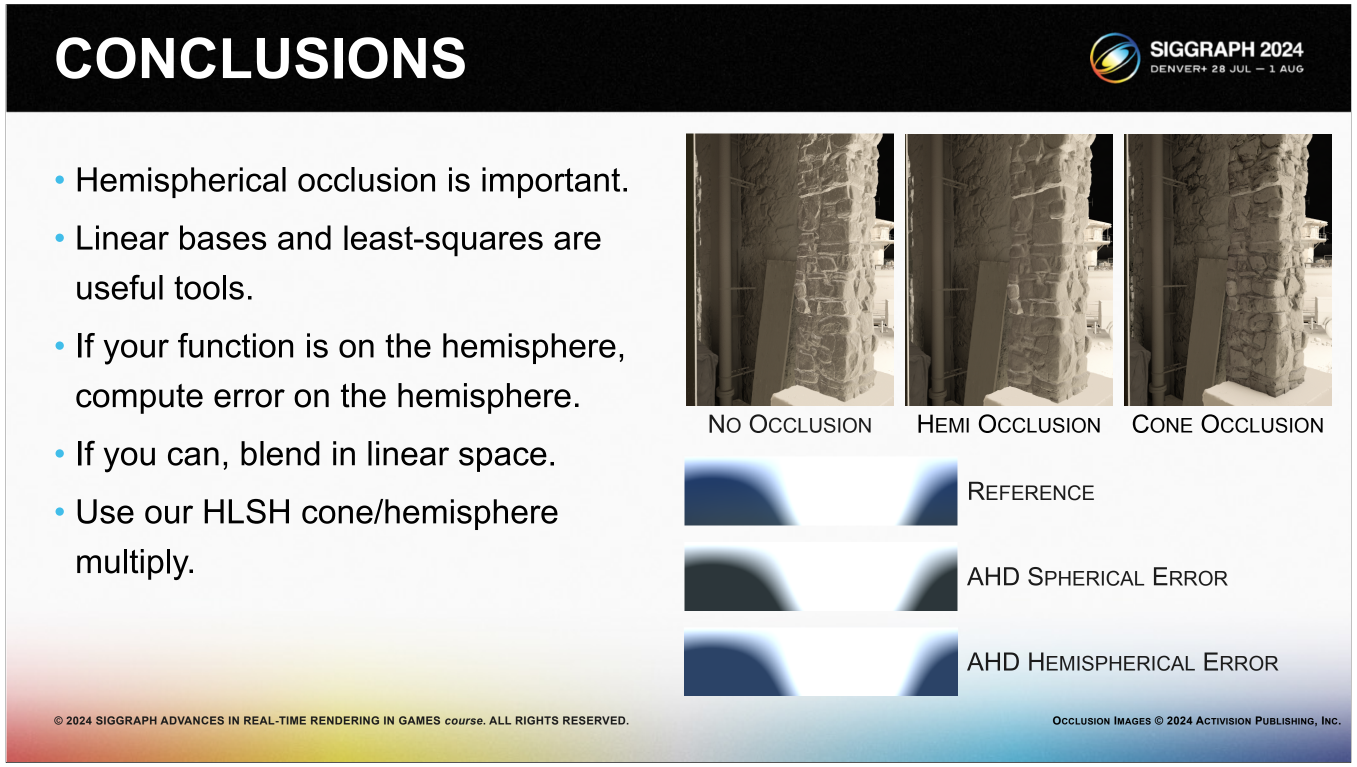
© 2024 Activision Publishing, Inc.

Here's the AO scale and the cone occlusion side by side on a different scene, where the lighting is coming from the top left. The AO scale mostly looks like corner darkening, as you'd expect from AO techniques, while the cone multiply has much better directionality.



© 2024 Activision Publishing, Inc.

And one final comparison. Look at the extra grounding shadows behind the monitor and crate, and, more subtly, the better integration of the back panel with the wall.



CONCLUSIONS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Hemispherical occlusion is important.
- Linear bases and least-squares are useful tools.
- If your function is on the hemisphere, compute error on the hemisphere.
- If you can, blend in linear space.
- Use our HLSH cone/hemisphere multiply.

NO OCCLUSION

HEMI OCCLUSION

CONE OCCLUSION

REFERENCE

AHD SPHERICAL ERROR

AHD HEMISPHERICAL ERROR

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

OCCLUSION IMAGES © 2024 ACTIVISION PUBLISHING, INC.

To wrap up, some key ideas I hope you take away from today.

- Firstly: hemispherical occlusion is important! You don't want your normal maps to glow.
- Linear bases and least-squares are useful tools. Most of the maths here is just linear algebra and some basic multivariate calculus, and it's easy to experiment with your own basis functions or derivations.
- If your function is on the hemisphere, compute error on the hemisphere.
- If you can, blend in linear space. It'll be much better behaved.
- Finally, if you're using spherical harmonics, use our HLSH cone or hemisphere multiply! It's an easy way to improve your visual quality, and all the code is up on ShaderToy.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

QUESTIONS

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.

Thank you for your time. I’ve got a few minutes for questions.

REFERENCES

[CAC+96] John Carmack, Michael Abrash, John Cash, et al., 1996. Quake.

[CDD+99] John Carmack, Robert A. Duffy, Jim Dosé, et al., 1999. Quake III: Arena.

[GKPB04] Pascal Gautron, Jaroslav Křivánek, Sumanta N Pattanaik, and Kadi Bouatouch. 2004. A Novel Hemispherical Basis for Accurate and Efficient Rendering. In *EGSR Proceedings*.

[Hable14] John Hable. 2014. Simple and Fast Spherical Harmonic Rotation. http://filmicworlds.com/blog/simple-and-fast-spherical-harmonic-rotation/.

[HW10] Ralf Habel and Michael Wimmer, 2010. Efficient Irradiance Normal Mapping. In *I3D Proceedings*.

[IS17] Michał Iwanicki and Peter-Pike Sloan, 2017. Ambient Dice. In *Eurographics Symposium on Rendering.*

[Ishmukhametov11] Denis Ishmukhametov, 2011. Efficient Irradiance Normal Mapping.

[Iwanicki13] Michał Iwanicki, 2013. Lighting Technology of The Last of Us. In *SIGGRAPH Talks.*

[JWPJ16] Jorge Jimenez, Xian-Chun Wu, Angelo Pesce, Adrian Jarabo, 2016. Practical Realtime Strategies for Accurate Indirect Occlusion. In *Activision Technical Memos.*

[Martin11] Sam Martin, 2011. Enlighten Real-Time Radiosity. In *SIGGRAPH Real-Time Live!*

[McTaggart04] Gary McTaggart, 2004. Half-Life 2 Source Shading. In *Game Developers Conference*.

[NP15] David Neubelt and Matt Pettineo, 2015. Advanced Lighting R&D at Ready At Dawn Studios. In: *SIGGRAPH Course: Physically-Based Shading in Theory and Practice.*

[Pettineo24] Matt Pettineo, 2024. The Baking Lab. https://github.com/TheRealMJP/BakingLab.

[RH01] Ravi Ramamoorthi and Pat Hanrahan, 2001. An Efficient Representation for Irradiance Environment Maps. In *SIGGRAPH Proceedings*.

[RSSIS24] Thomas Roughton, Peter-Pike Sloan, Ari Silvennoinen, Michał Iwanicki, and Peter Shirley, 2024. ZH3: Quadratic Zonal Harmonics. In *I3D Proceedings.*

[RSSS24] Thomas Roughton, Peter-Pike Sloan, Ari Silvennoinen, and Peter Shirley, 2024. Hemispherical Lighting Insights. In *Activision Technical Memos.*

[SLS05] Peter-Pike Sloan, Ben Luna, and John Snyder, 2005. Local, Deformable Precomputed Radiance Transfer. *ACM Trans. Graph.* 24, 3.

[Snyder96] John Snyder, 1996. Area Light Sources for Real-Time Graphics. *Microsoft Tech Report.*

[SS18] Peter-Pike Sloan and Ari Silvennoinen, 2018. Directional Lightmap Encoding Insights. In *SIGGRAPH Asia Technical Briefs.*

[SS20] Peter-Pike Sloan and Ari Silvennoinen, 2020. Precomputed Lighting Advances in Call of Duty: Modern Warfare. In *SIGGRAPH 2020 Advances in Real-Time Rendering*.

Environment Maps:

- Autumn Field: Sergej Majborada, 2024. https://polyhaven.com/a/autumn_field
- Evening Road: Sergej Majborada, 2020. https://polyhaven.com/a/evening_road_01
- Vienna Garage, Hallstatt, Linz, Wells: Bernhard Vogl, 2010. http://dativ.at/lightprobes/

© 2024 SIGGRAPH ADVANCES IN REAL-TIME RENDERING IN GAMES *course*. ALL RIGHTS RESERVED.