Advanced Lighting R&D at Ready At Dawn Studios

David Neubelt – Ready At Dawn Studios

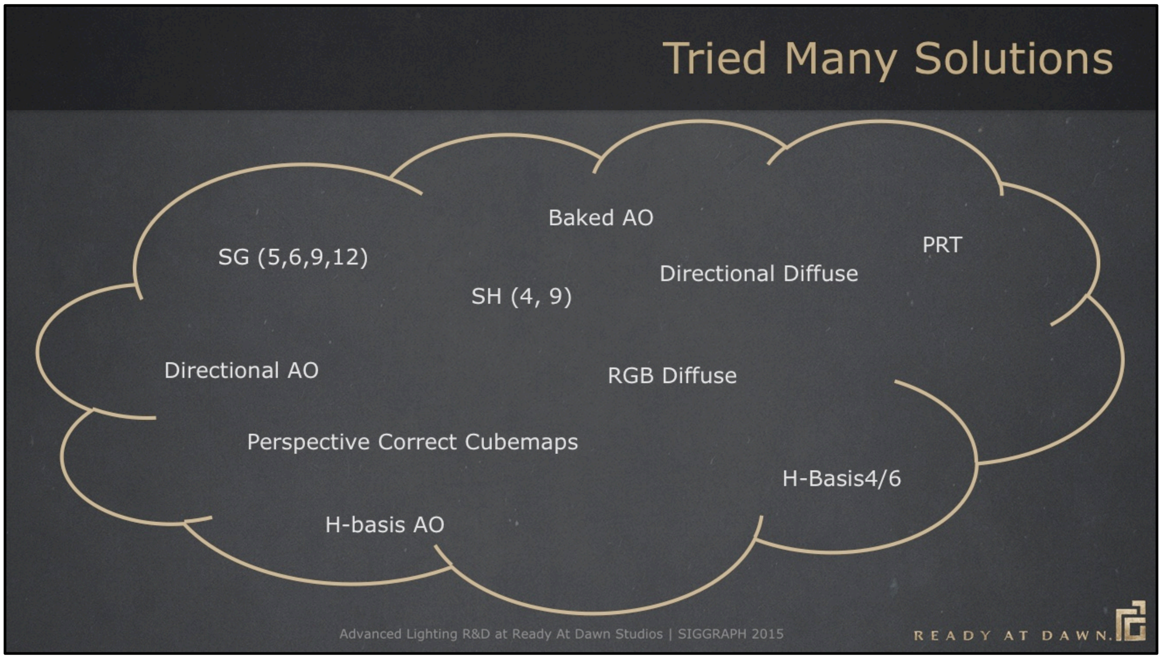
Matt Pettineo – Ready At Dawn Studios

Physically Based Shading in Theory and Practice 2015

This screen shot is one of the best examples of our baked GI technique because it shows complicated lighting across a range of materials and it uses no runtime lighting. Everything is based off our baked GI solution.

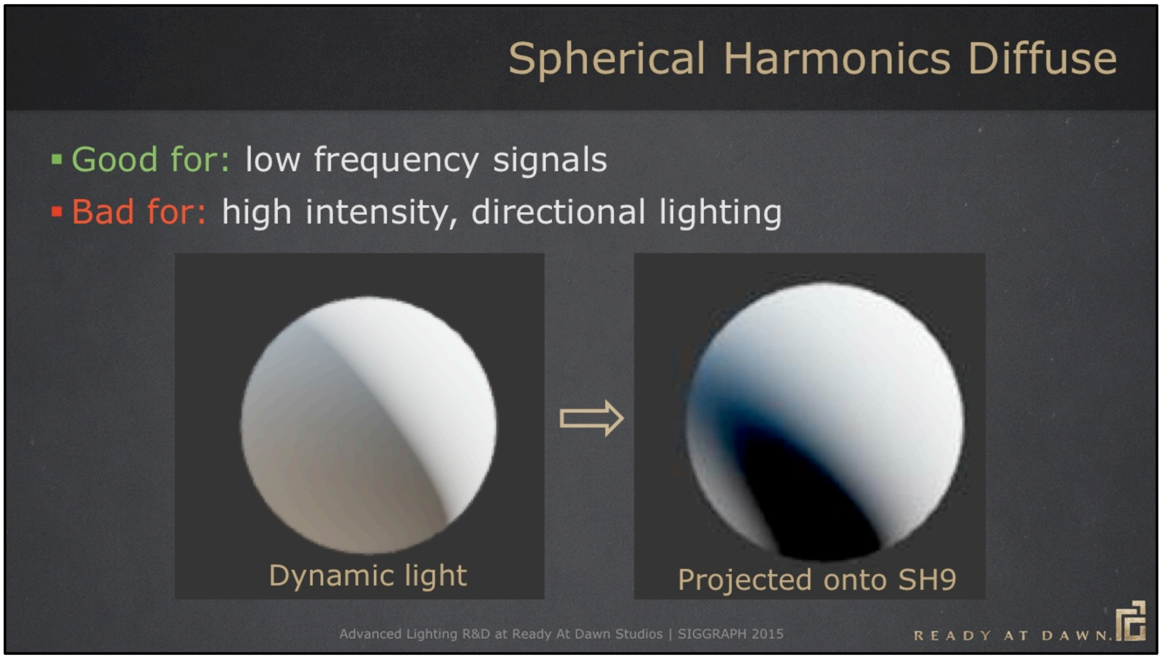
The Order: 1886 is a linear game with medium size levels both inside and outside. The game was fairly static in it's lighting so it made the most sense for our project to use a statically baked GI solution.

It was important for us to not only capture diffuse lighting but also capture specular GI as well. One of the many reasons we pushed so hard on the baked specular was the limitations we hit using cubemaps (spatially getting wrong reflections in the wrong places and the angular quality).



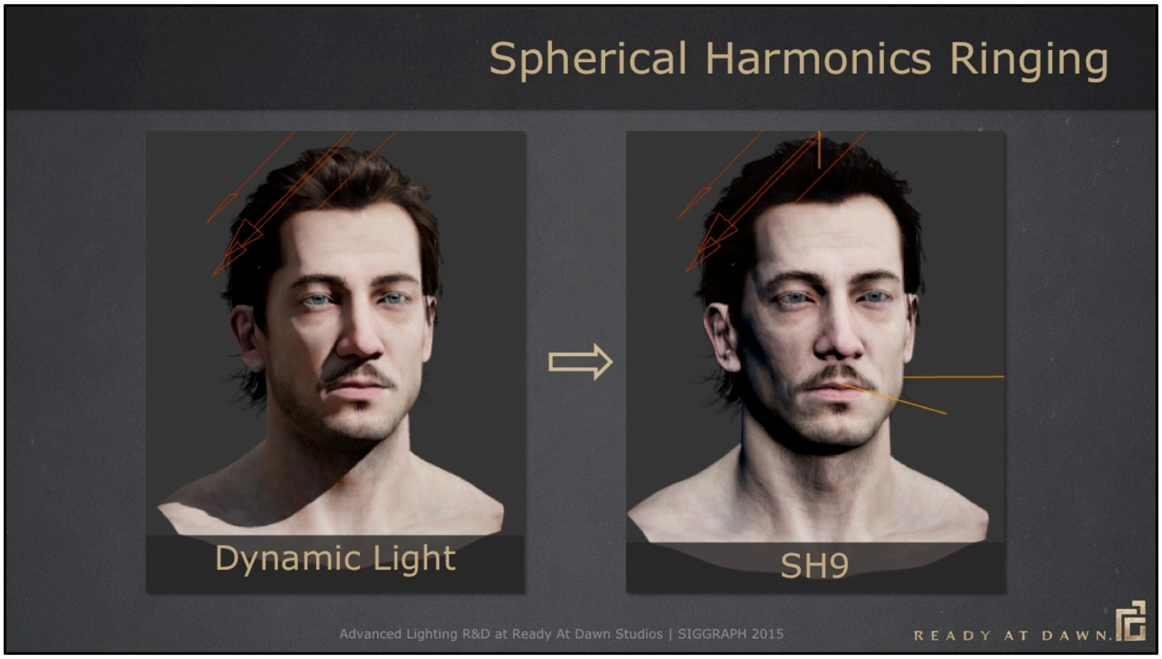
We tried many techniques during the development and each of them didn't really fit our needs exactly. Some techniques even stayed into the shipping product like the H-basis AO for specular and diffuse occlusion for prop objects.

However, what we ended up using was Spherical Gaussians and we'll quickly go into why we made that decision.

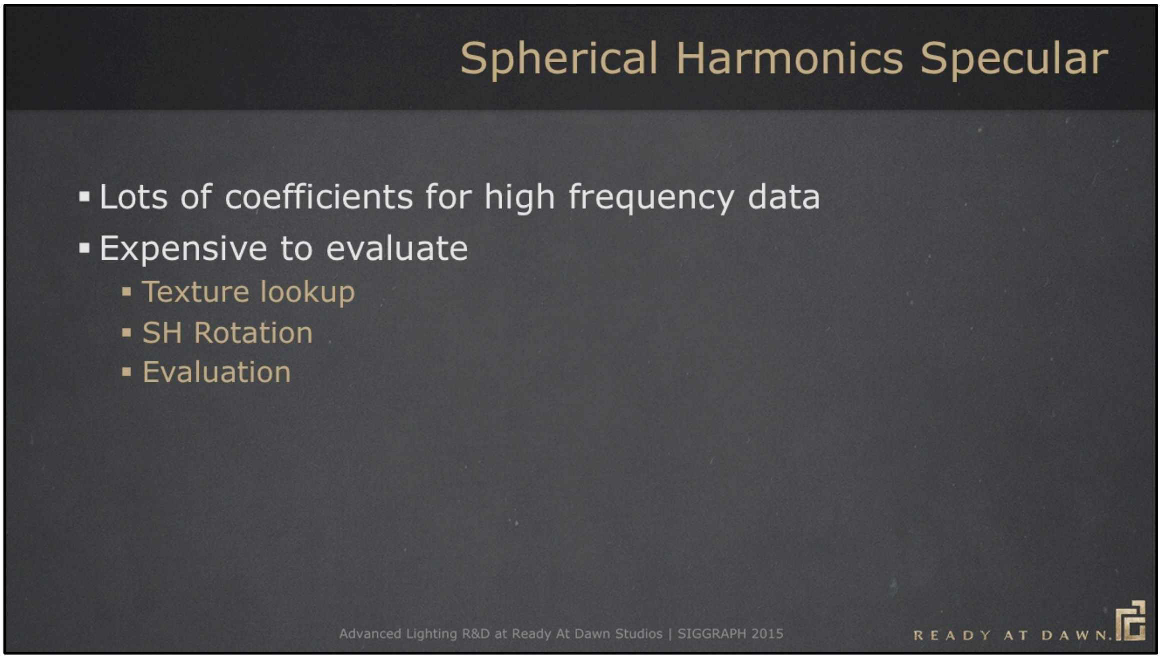


Note: You can apply a windowing function to reduce ringing at the cost of blurring the signal.

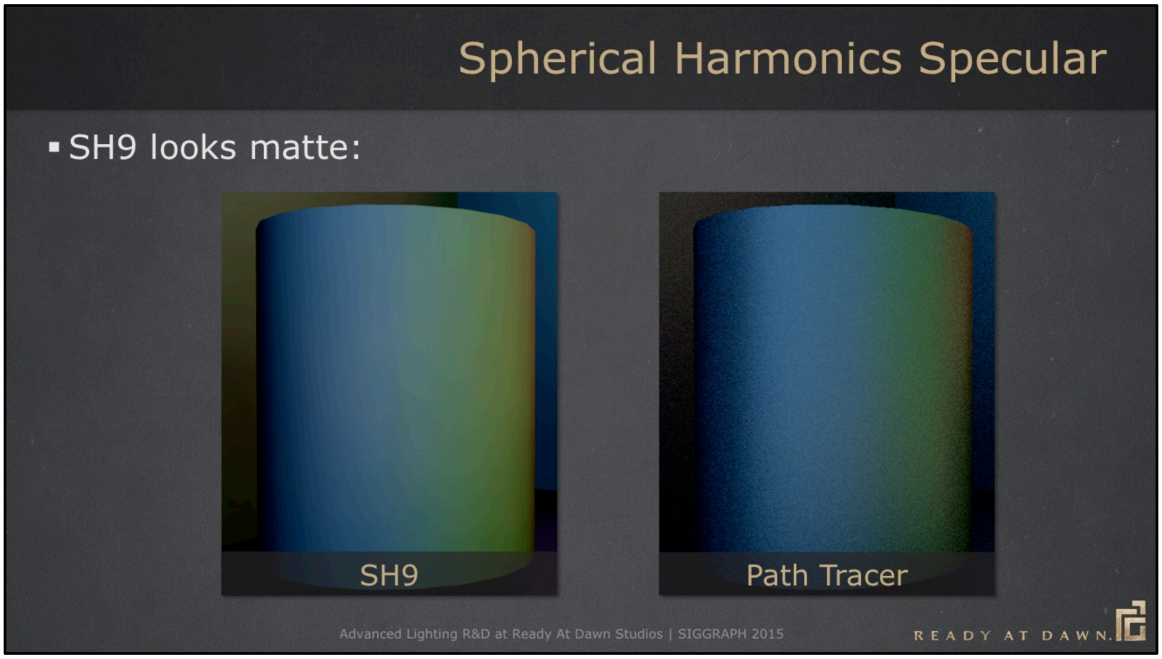
HDR lighting can cause certain lobes to be very large negative numbers to cancel out the high positive coefficients. Bad for quality and bad for compression.



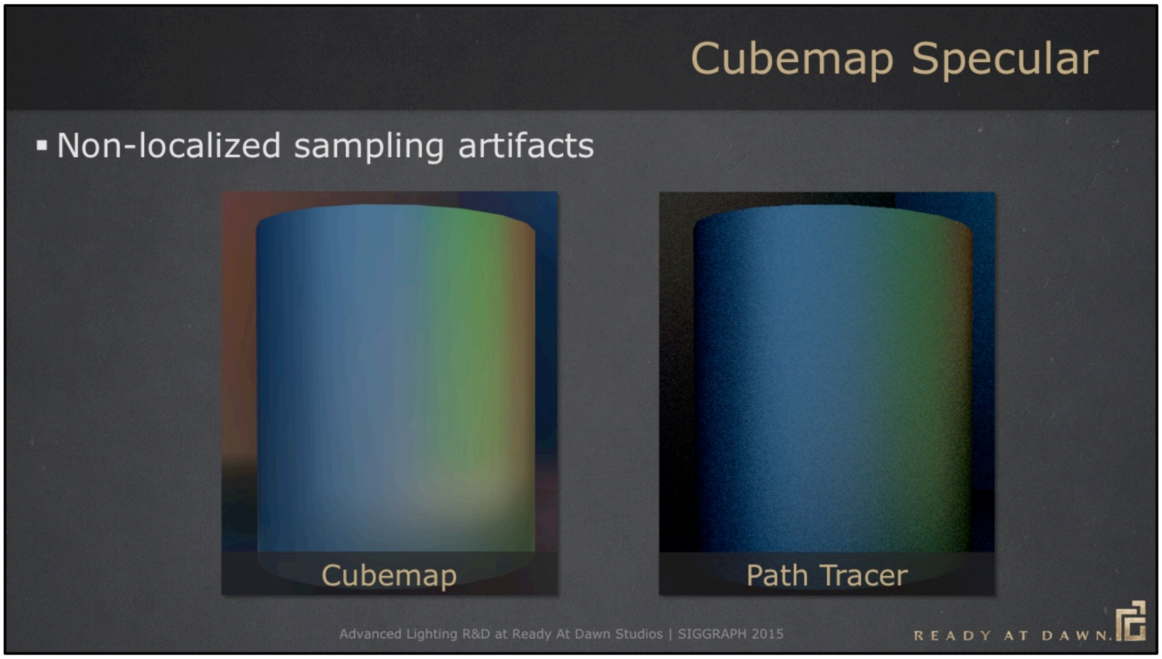
Ringing showed up in our game when we baked the direct lights to save on lighting costs. This caused our characters to have a dead/zombie look. The ringing caused the lighting to overshoot, e.g. on his forehead, and go negative on the opposite side of the lighting.



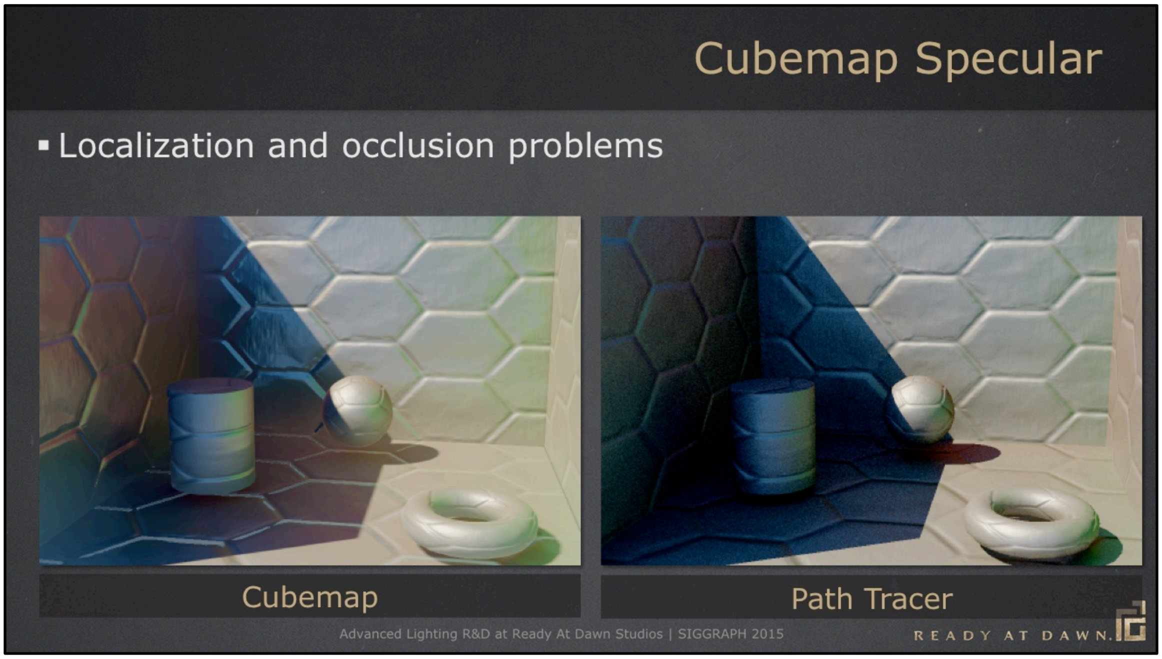
We tried spherical harmonics specular that Bungie had implemented. It was a little too expensive because of the combination of the lookup, sh rotation, and brdf evaluation.



It also had a matte look with only 9 coefficients. It might have been good enough with 16 coefficients but the cost certainly would have been too expensive at that point.

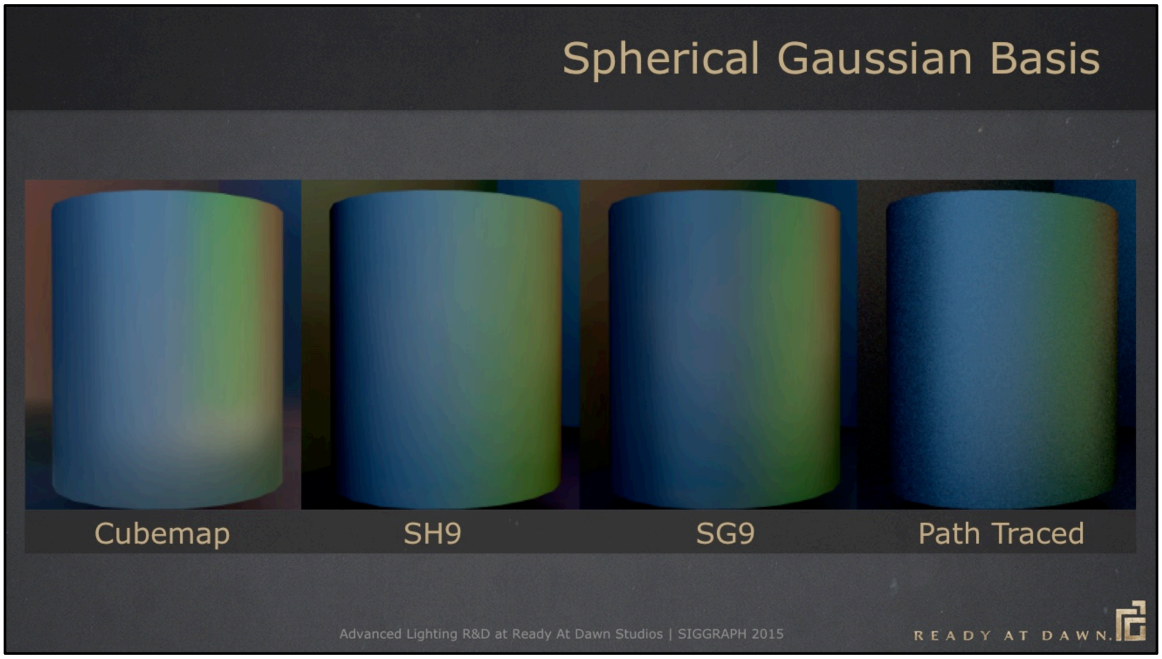


- Matches lighting signal and BRDF more closely
- Non-localized
    - Implemented AABB perspective warping
    - Still not perfect and hard to author for non-square rooms
- Causes glowing edges
    - Lack of occlusion



The screenshot on the left was a very common problem for our game. Grabbing the wrong specular probe would cause glowing edges in the dark or not enough energy.

We tried baking directional occlusion into an h-basis and let the artists specify a distance but it was difficult and error prone. It also didn't help when you wanted specular and you weren't getting it because the cubemap was in the wrong place. The artists spent a lot of time tweaking cubemap locations.

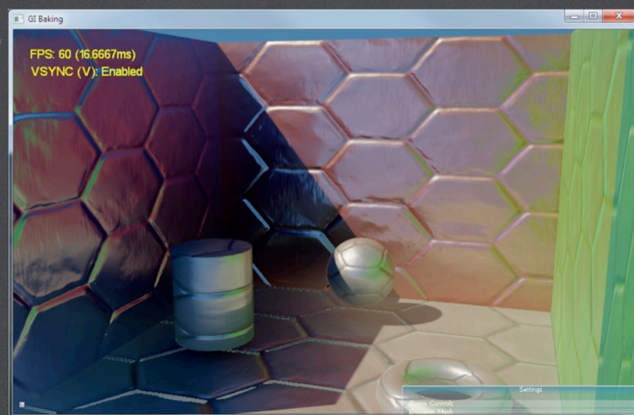


Here is a comparison of the 3 main techniques. For spherical Gaussians notice the width of the highlights matches the path traced solution the closest.

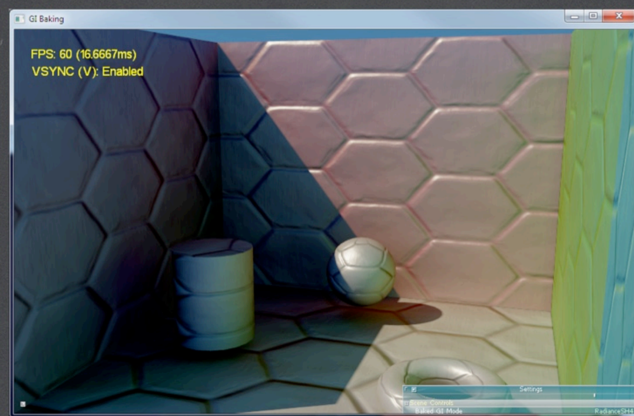
Flat Ambient

Single Scalar Diffuse

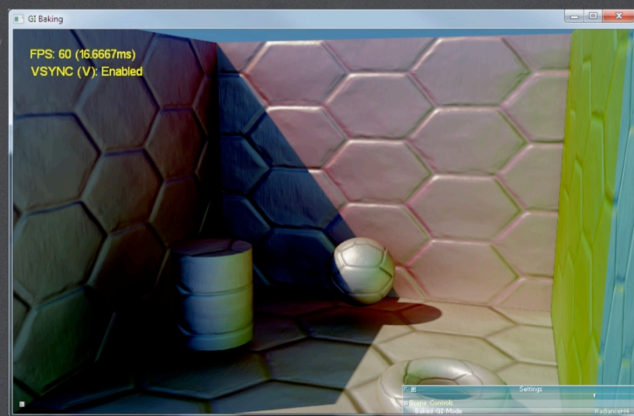
Cubemap



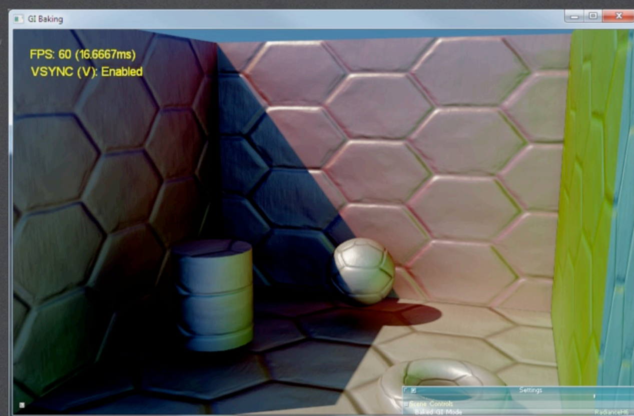
Spherical Harmonics 4



H-Basis 4



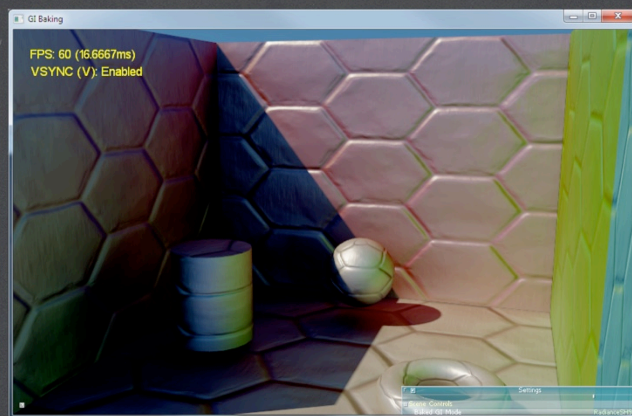
H-Basis 6



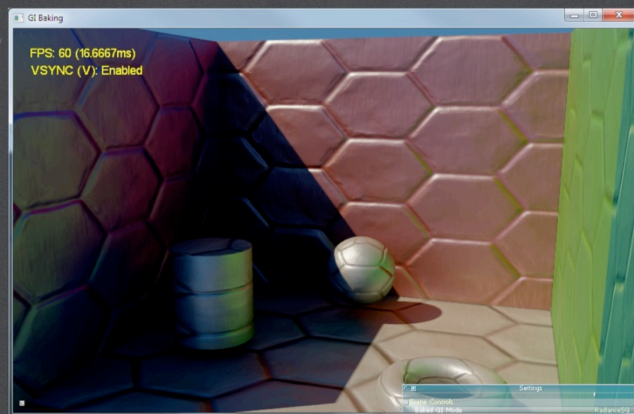
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015

READY AT DAWN.

Spherical Harmonics 9



Spherical Gaussians 9

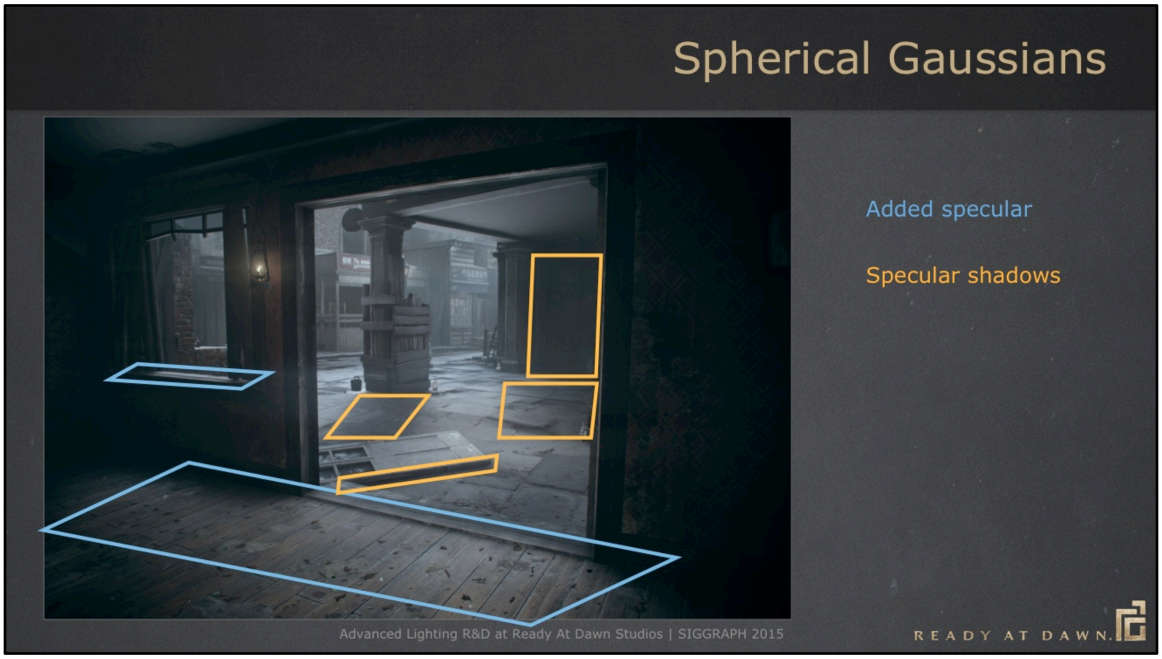


Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015

READY AT DAWN.

We'll now go over some motivating comparisons of SG bakes.

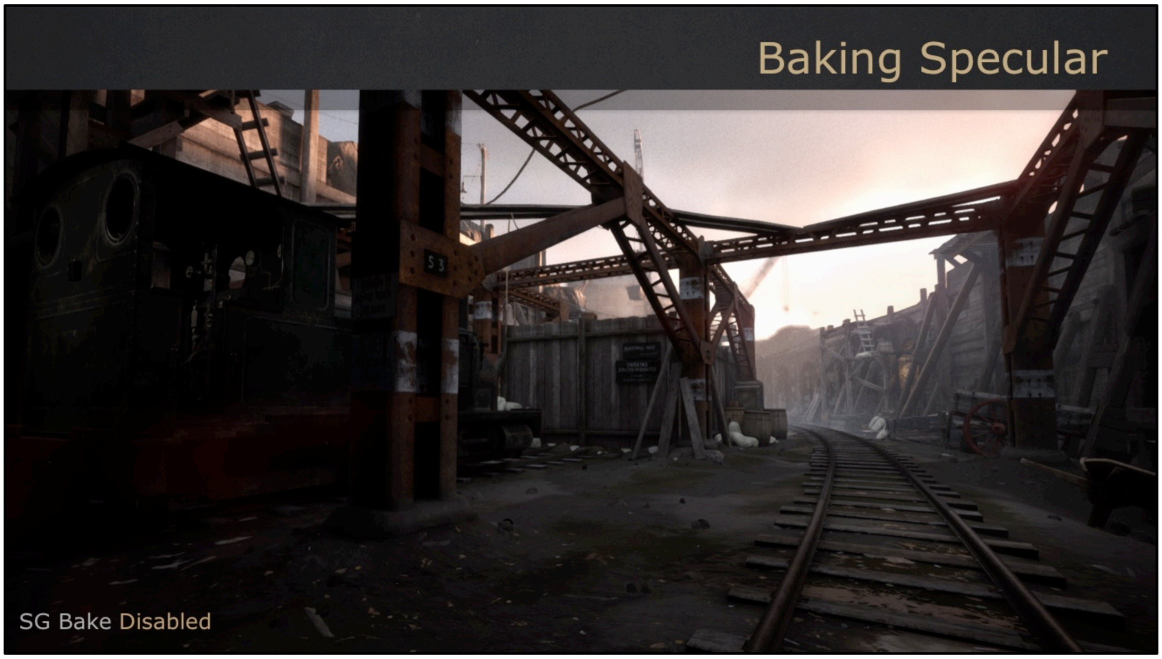
This is an old screenshot showing diffuse baked lighting with cubemaps. The energy looks flat and floor is getting energy where it shouldn't.



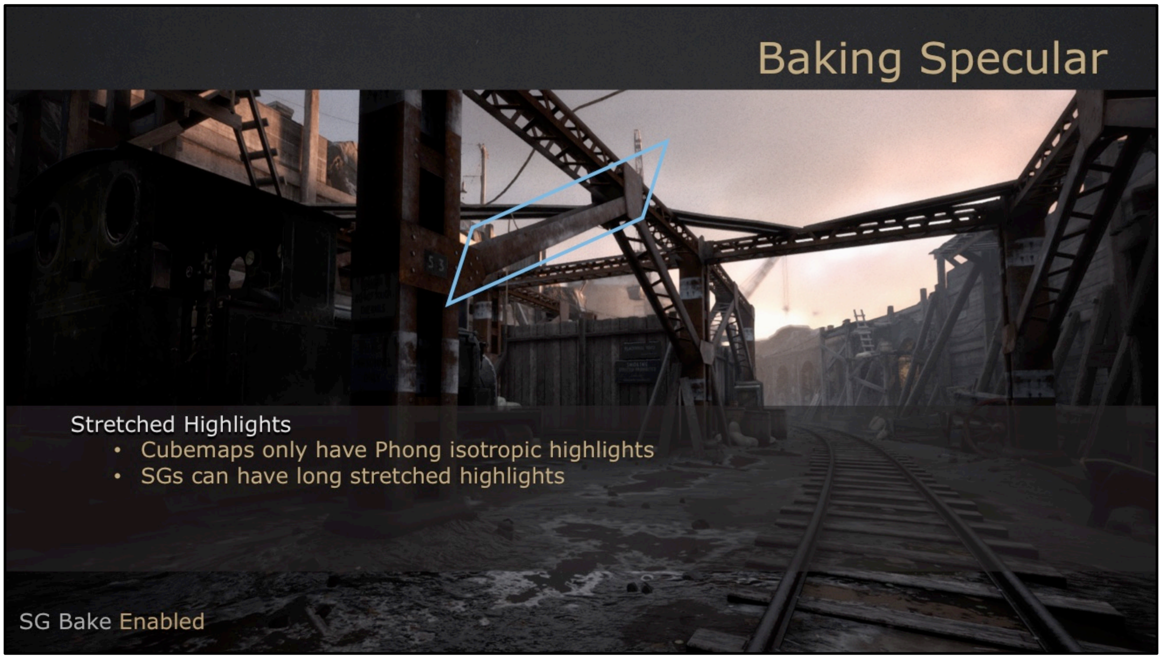
In this example, we re baked using spherical gaussians. Notice the light is where it should be and the shadows are also where they should be. The scene looks a lot more interesting now because of the mixture of shadows and highlights.

In this scene we disabled the Spherical Gaussian lighting.

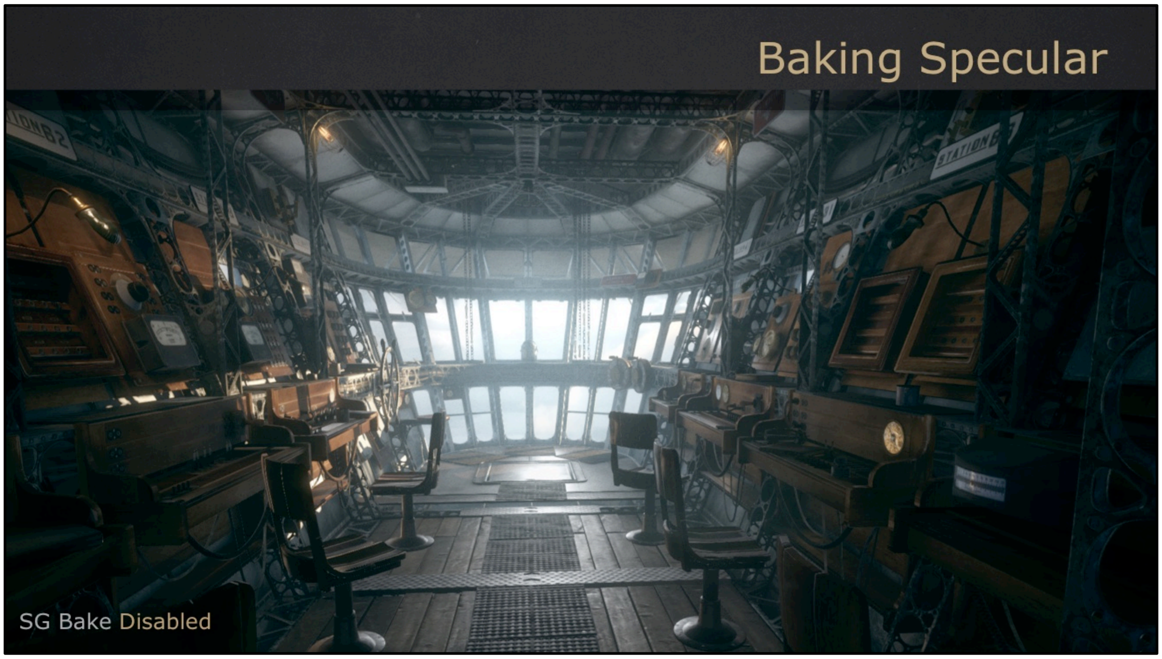
When its re-enabled notice that the hot spots around the light. It would be very difficult to capture this type of lighting with cubemaps because you'd need to place many cubemaps around each light source.



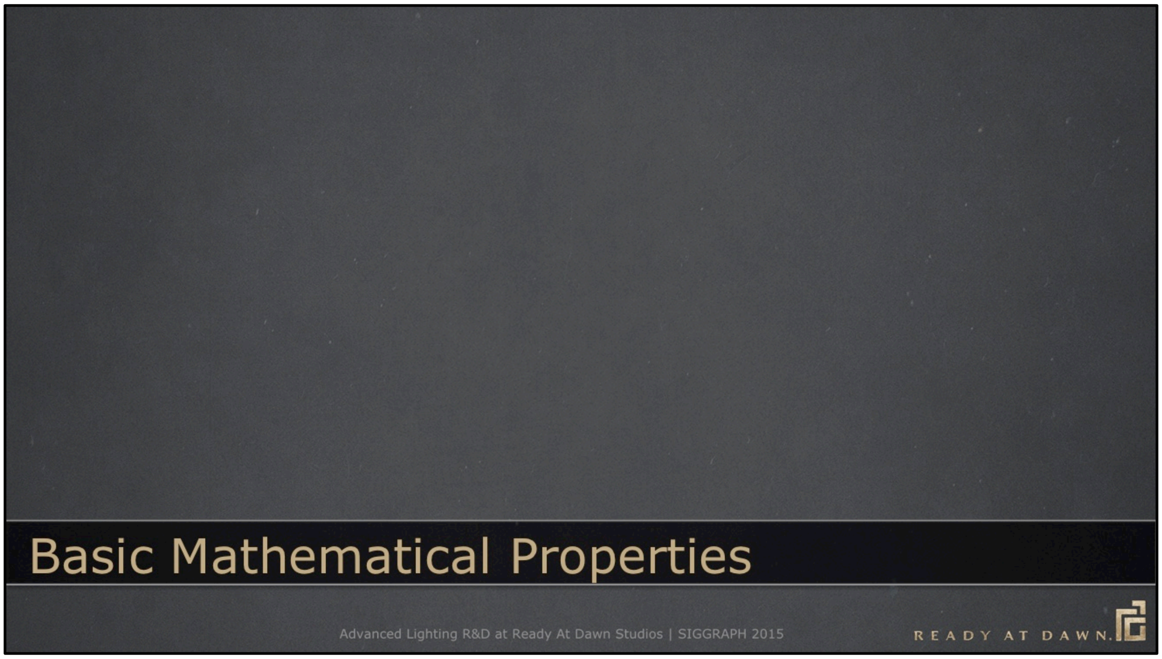
This scene is very dark – it's missing a lot of specular. In this example the artists didn't have enough cube maps. Had they put a cubemap in this area then it probably would have looked a bit better. However, they didn't because of memory constraints for this level. The level was very large and had a huge amount of gameplay area so getting enough cubemaps in every spot wasn't a viable option.

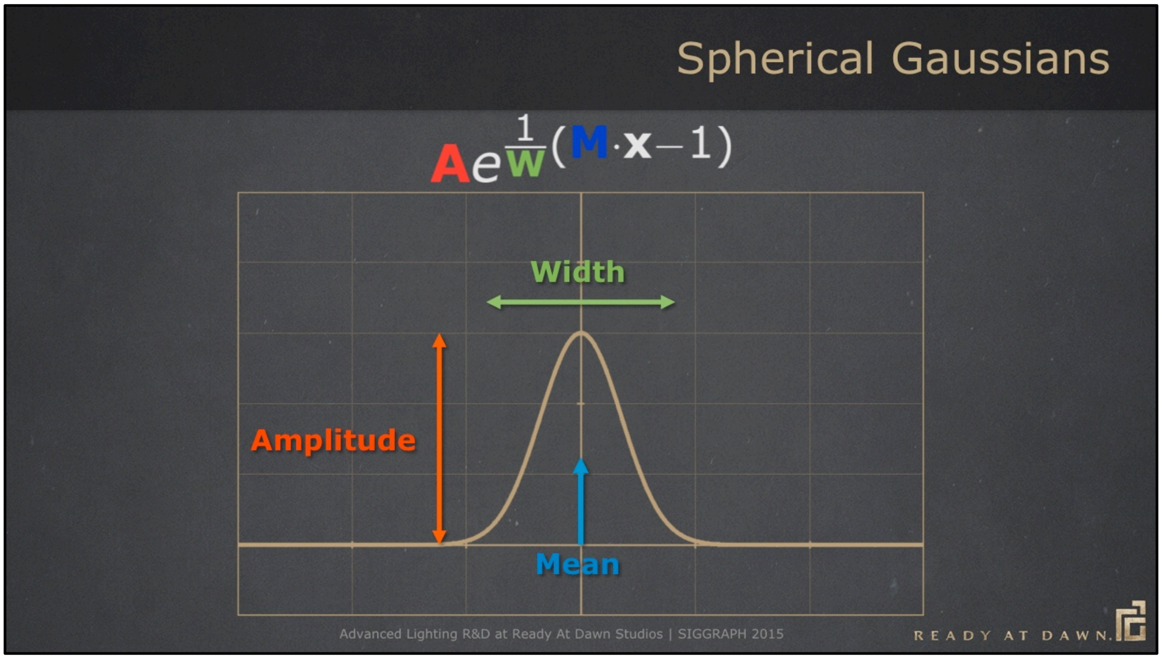


Notice when we enable SG's the energy in the area comes back and you also get the stretched highlight on the metal beam. Spherical Gaussian's use the half angle parameterization for BRDFs which allows for elongated highlights while cubemaps only support an circular highlights.


Baking Specular
SG Bake Disabled

In this example we see two items. We see the elongated highlights on the floor and wood panels. We also notice all the complicated geometry if we had used cubemaps in this area only all of the geometry would have been glowing bright.


Basic Mathematical Properties
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015
READY AT DAWN.



In my opinion, SG's are much easier to understand because they have intuitive parameters.

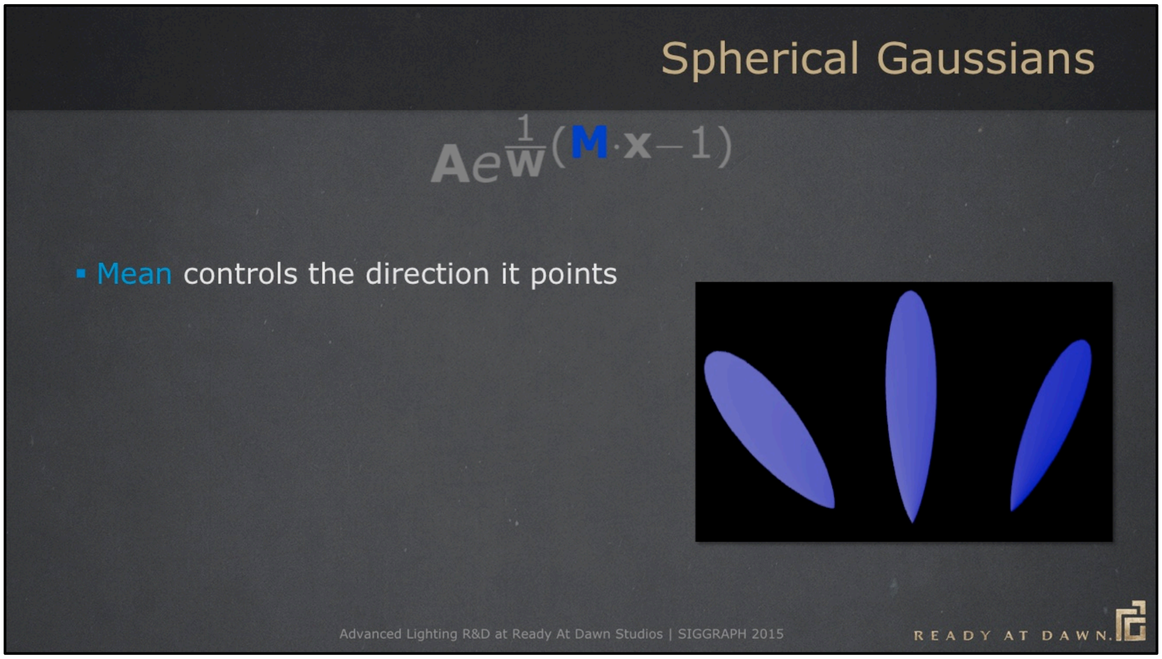
They also can be graphed in 2d because they are isotropic. In this visualization I'm taking a slice along the middle and the x-axis represents the angle away from the mean.

Its an exponential Gaussian like function that falloffs based on the geodesic distance from the mean.

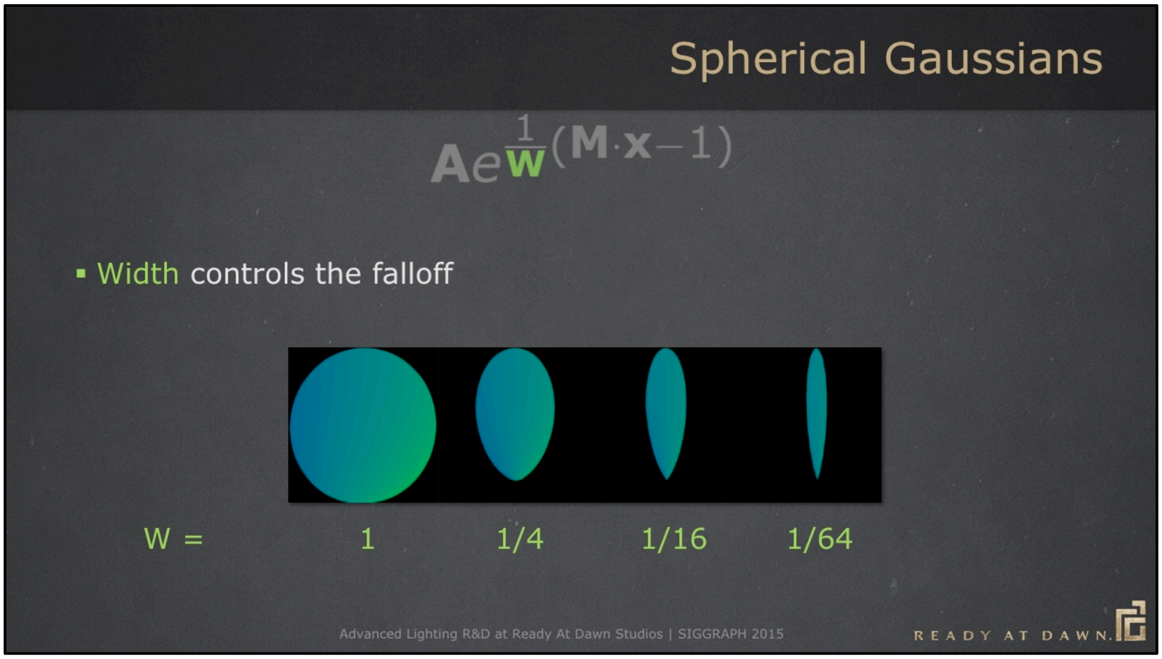
Spherical Gaussians

$$\mathbf{A}e^{\frac{1}{\mathbf{w}}(\mathbf{M}\cdot\mathbf{x}-1)}$$

- Mean controls the direction it points
- Width controls the falloff
- Amplitude controls the height or intensity

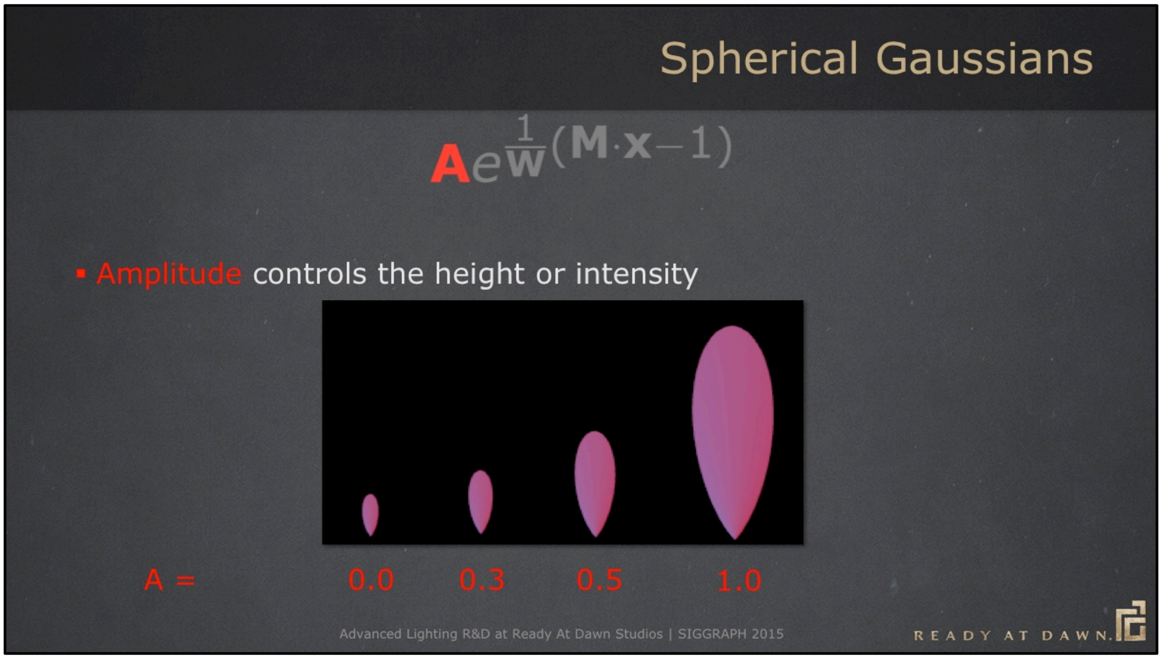


The mean is simple a normalized directional vector. It’s easy to rotate it to the direction you want.



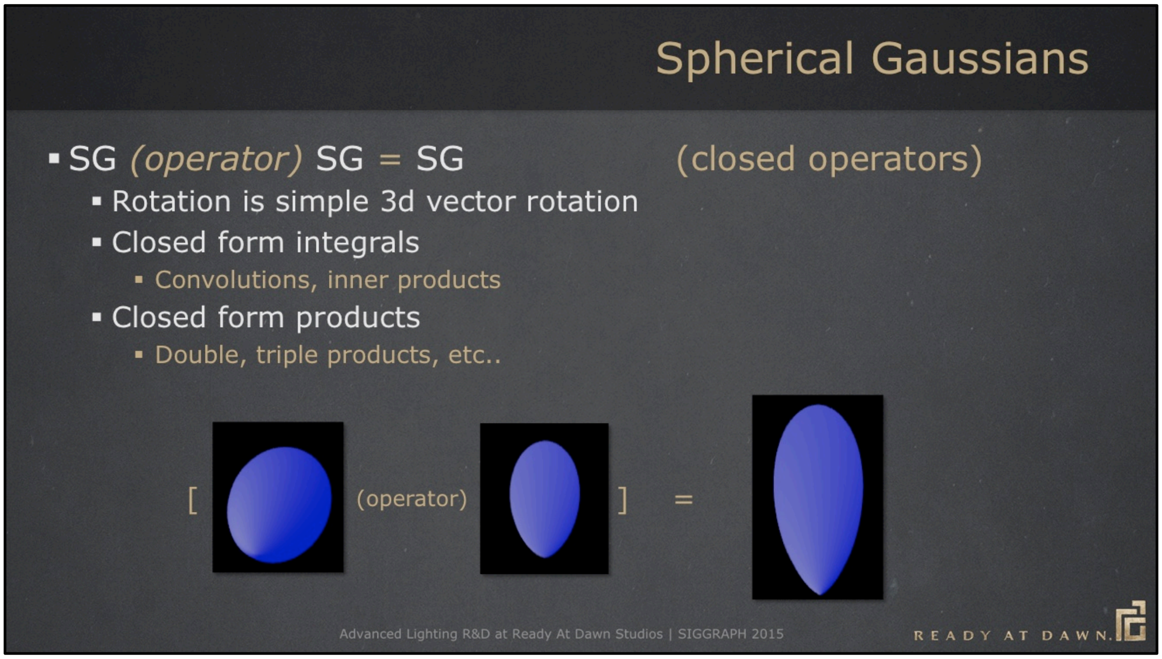
The width parameter or sometimes called falloff, frequency, or inverse width controls how quickly the function falls off from the mean.

Changing the width parameter changes the total amount of energy the function has.



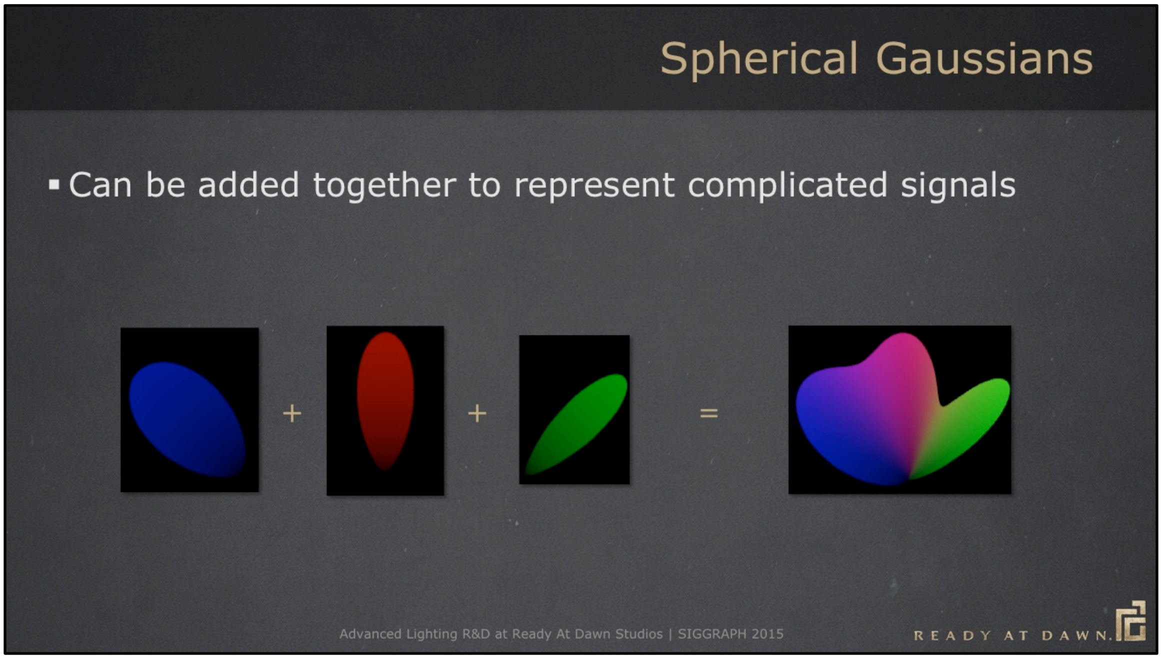
The amplitude controls the height. Its one of the more intuitive parameters because it has a linear response. For example, if you multiply the function by 2 then it will have twice the height at the mean direction.

However, note changing the height also will change the total energy of the function. Thus, amplitude and frequency are tied together for the total energy of the function.



The real power from SGs come from how many operators and closed form operators it supports. Because most of the operators generate another SG you can chain together many simplifications and get a single SG out.

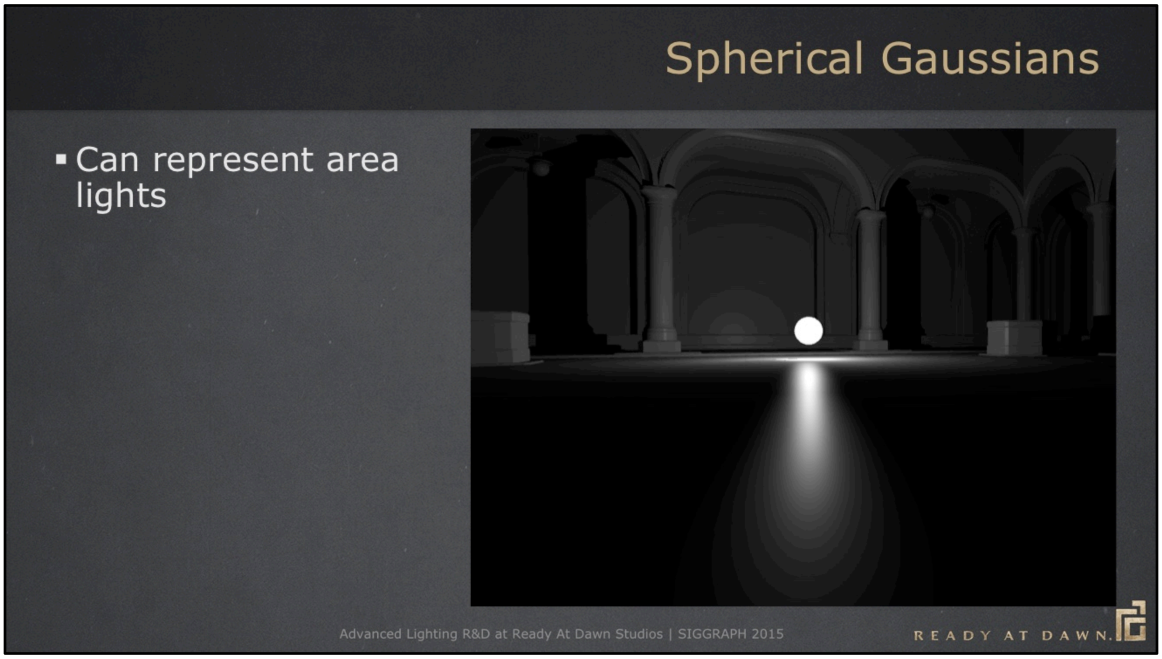
This allows us to take a complicated equation, like the rendering equation, and evaluate it analytically.



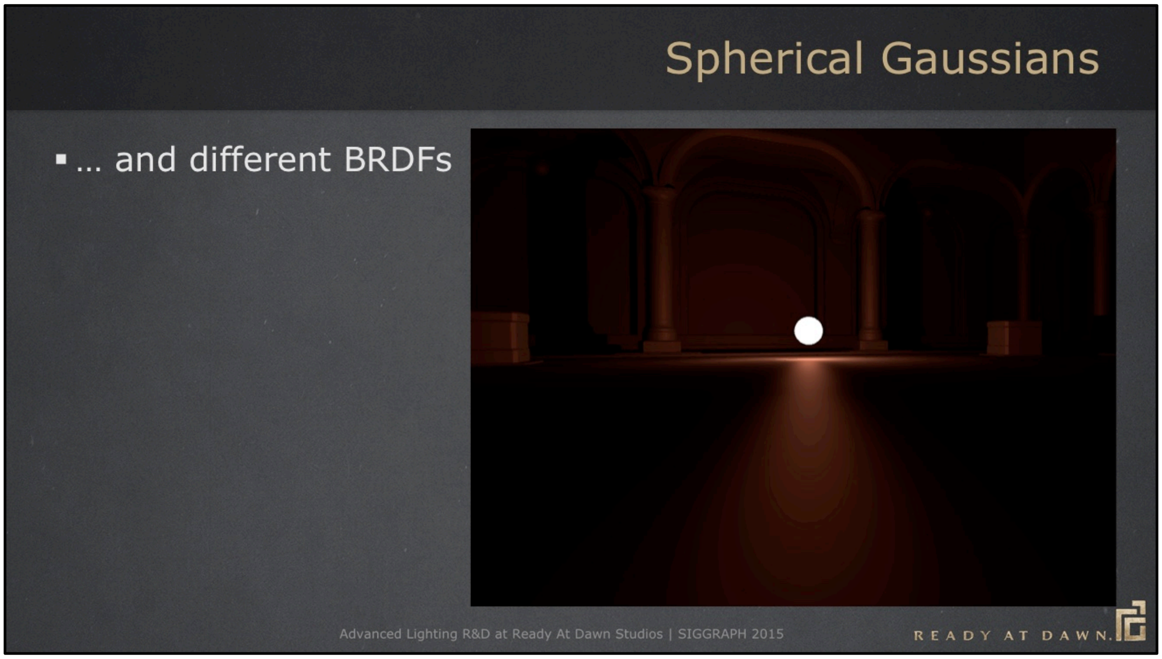
Another really interesting operation you can do with SGs is add them together.

Imagine you had a room with a blue wall on the left, red ceiling and a green wall on the right. If you could represent the radiance from each direction onto a SG like I did in the picture above then you can add them together to get the function on the right.

Simplying sampling the function on the right will give you the radiance in that direction.

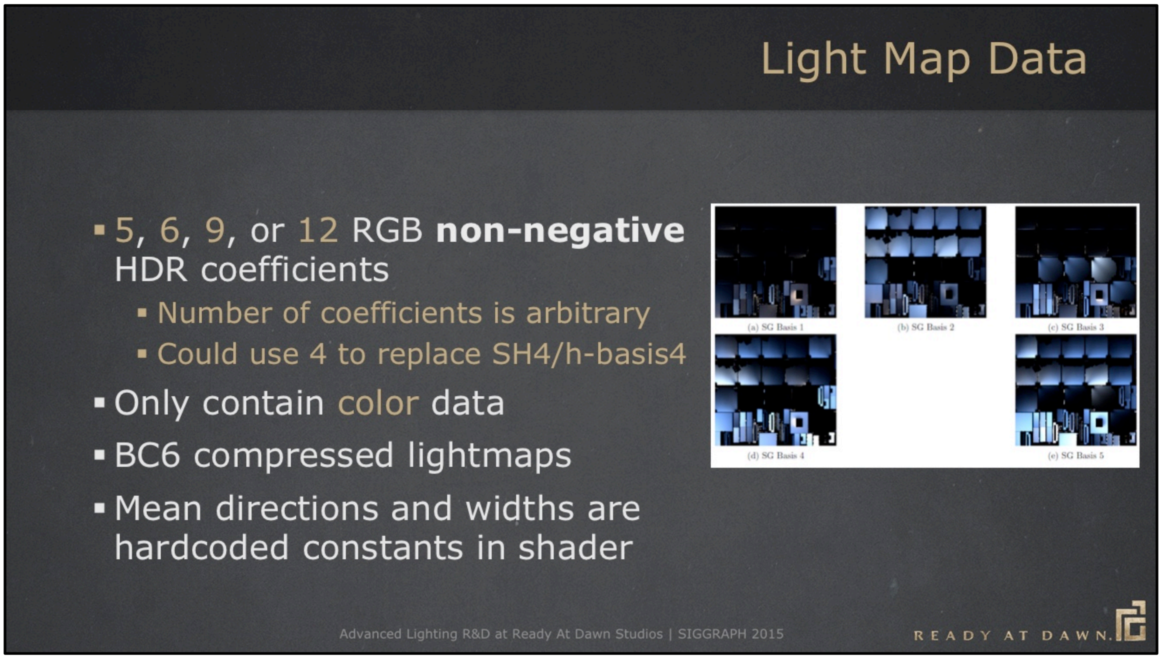


SGs can represent area lights.



SGs can represent BRDFs

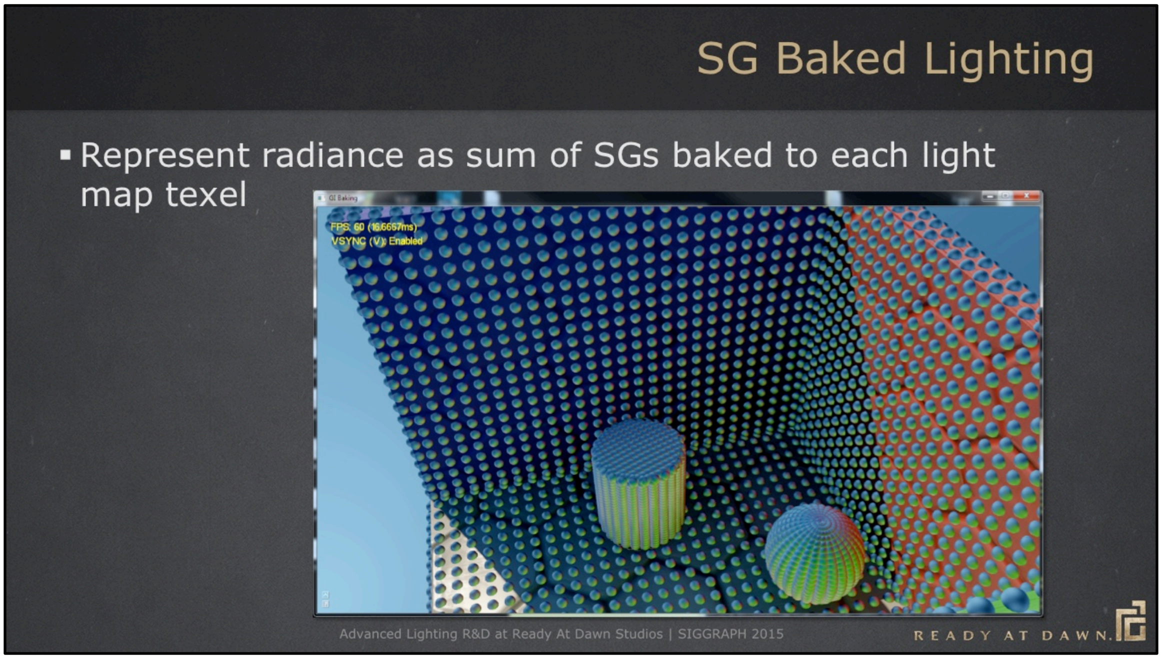
SGs in our game were stored into each texel of our lightmaps.



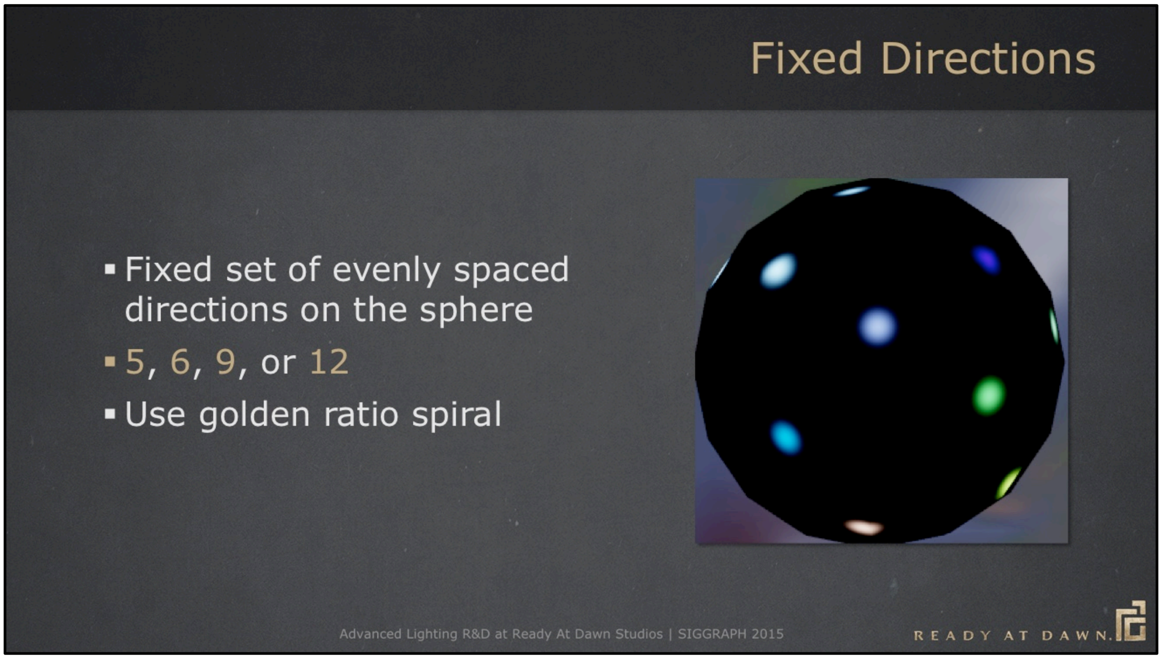
The only data stored in the lightmap is the HDR RGB color. The number of lobes you have will dictate the number of lightmaps you will have. It was very easy and intuitive for the artists to understand this concept. Since the lightmaps represent the radiance in the lobes direction it was also very easy to inspect and understand what was going on with each lightmap unlike SH.

The coefficients were only positive which helped out with compression and precision.

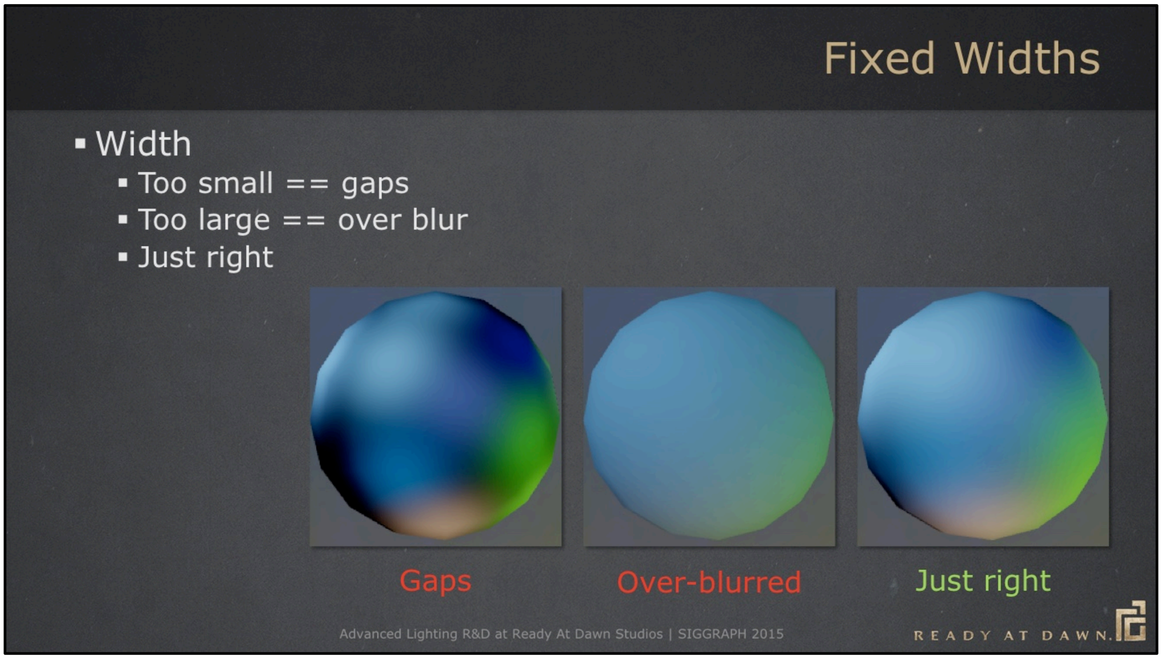
The directions or widths were not stored in the lightmap.



Here is a visualization of spherical gaussians on each texel in the lightmap. Each hemisphere is the sum of the # of SGs. As you can see the sphere in the picture shows the red radiance coming from the wall on the right, the sky on the top and the green wall behind the camera and the floor.



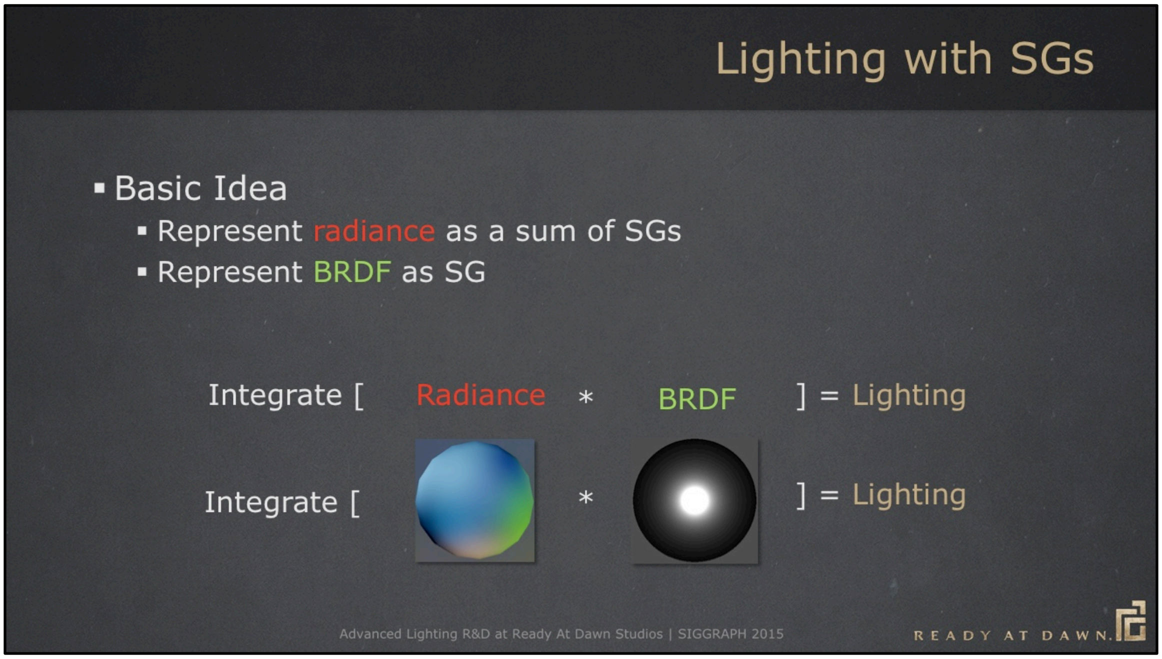
Based on the number of lobes we'd generate a uniform distribution of directions using the golden ratio spiral algorithm.



The width needed to be solved for so it was continuous, didn't have gaps, and wasn't too wide that it overblurred the signal.

Why Fixed Basis?

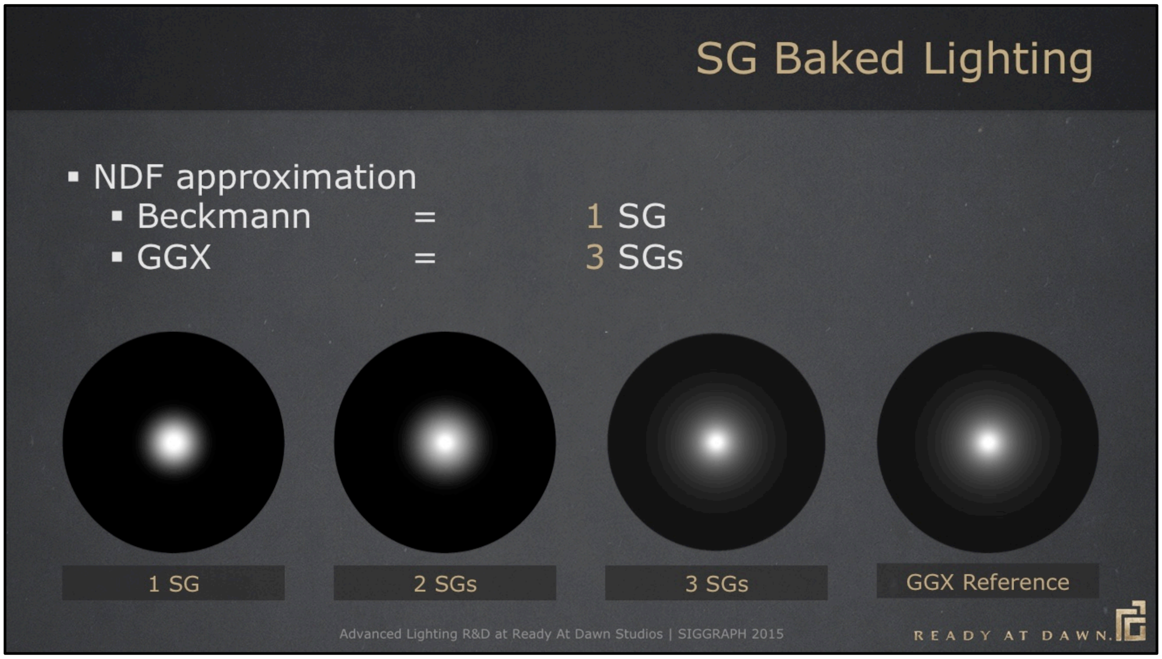
- Variable scalar count
  - 27 scalars for color (9 * 3)
  - 9 widths
  - 18 scalars for direction (theta/phi)
  - == 54
- Fixed scalar count
  - == 27
- Fixed Benefits
  - 2x more directions for same amount of data
  - More optimization opportunities
  - No interpolation issues between adjacent texels



If we can represent the radiance as a sum of SGs and the BRDF as a SG then we can use the analytical integration operator of SGs to get the lighting out. That's the basic idea.

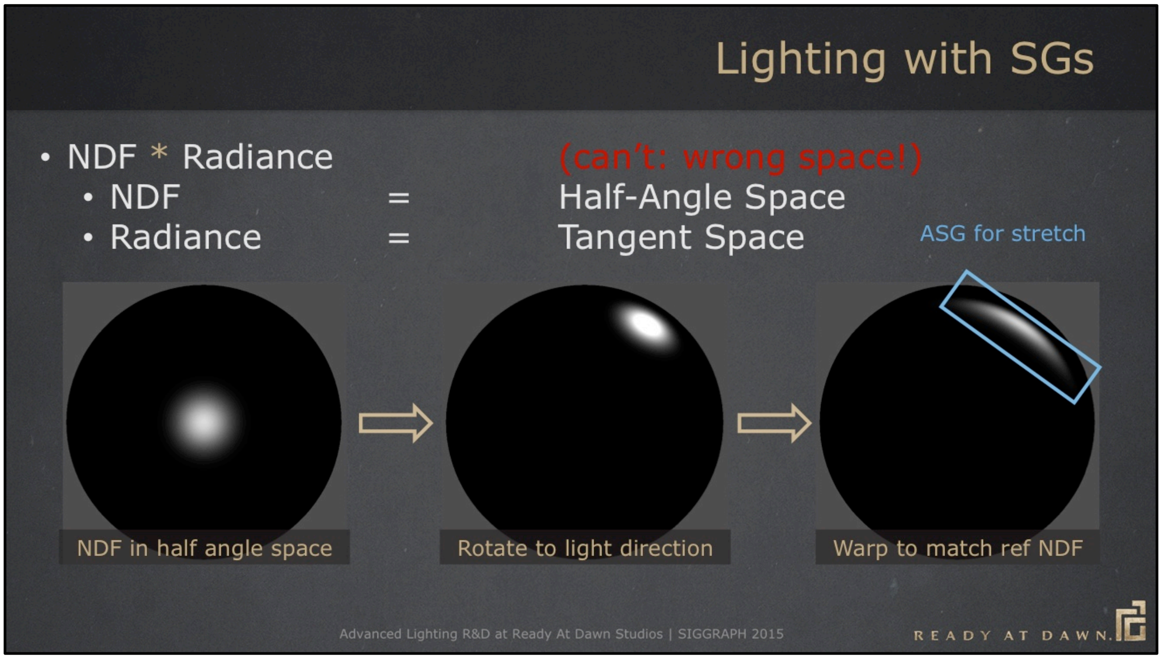
SG Baked Lighting

- Actual implementation is more complicated
  - Fresnel
  - Shadowing
  - Masking
  - Warping the NDF
    - NDF SG is parameterized by half angle
    - Radiance SG is parameterized by light direction
    - Need them in same space for efficient SG multiplication



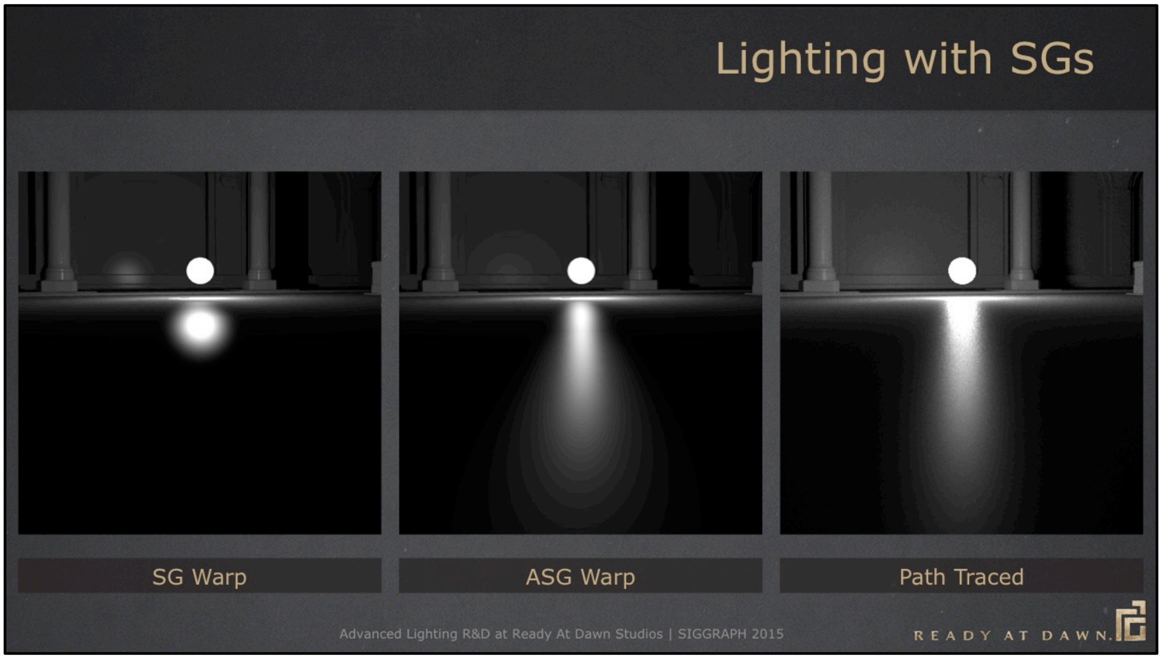
Approximating the Beckmann NDF can be done extremely accurately with only 1 SG. Approximating the GGX BRDF requires 3 SGs to get the nice wide tail.

For the order we could only afford 1 SG. However, with ambient lighting the tail of the GGX lobe isn't as noticeable as say it would be with a punctual light source so it wasn't as bad as it seems in this slide.

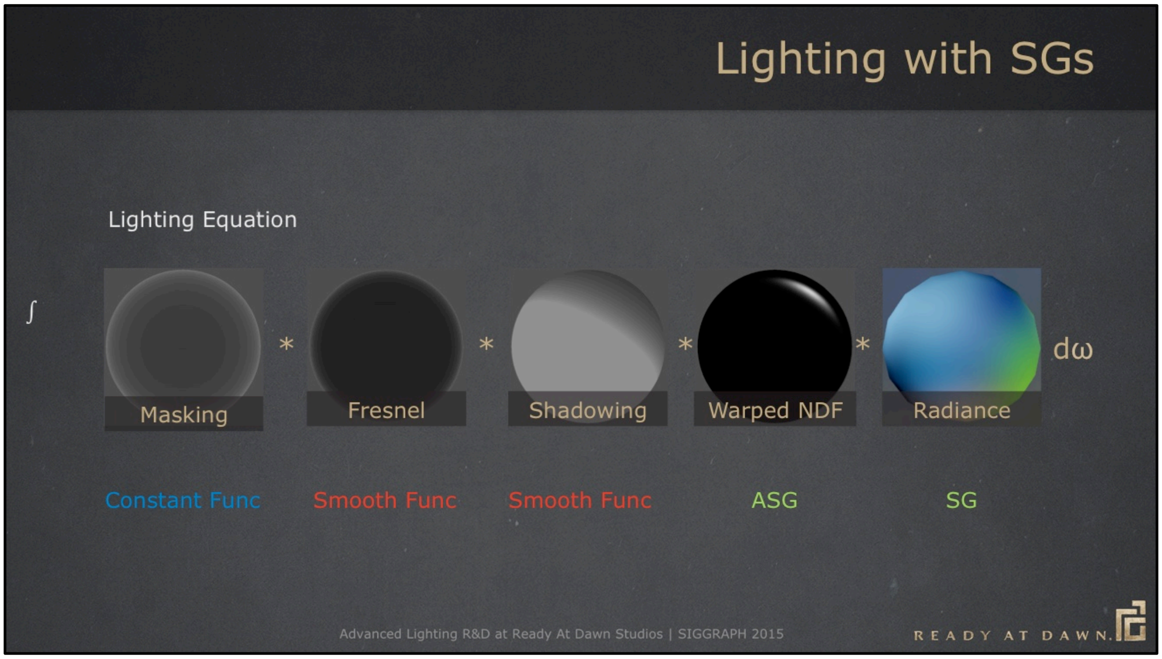


Unfortunately, we can't analytically integrate the radiance and the NDF because the input parameters to the SGs are in different spaces. If we were to integrate over the lighting directions then we have to convert that lighting direction to the half angle direction for the NDF. The main idea is to generate another SG that points in the same direction and best represents the NDF in that space.

The idea is we first rotate the NDF to point in the direction of the light direction. Then we modify the width to best represent the energy of the NDF. However, SG's can only modify width symmetrically but because of the half angle parameterization it actually scales in one direction more than the other. Because of this we promote the SG to an ASG(Asymmetric Spherical Gaussian). This allows us to stretch on two directions independently.



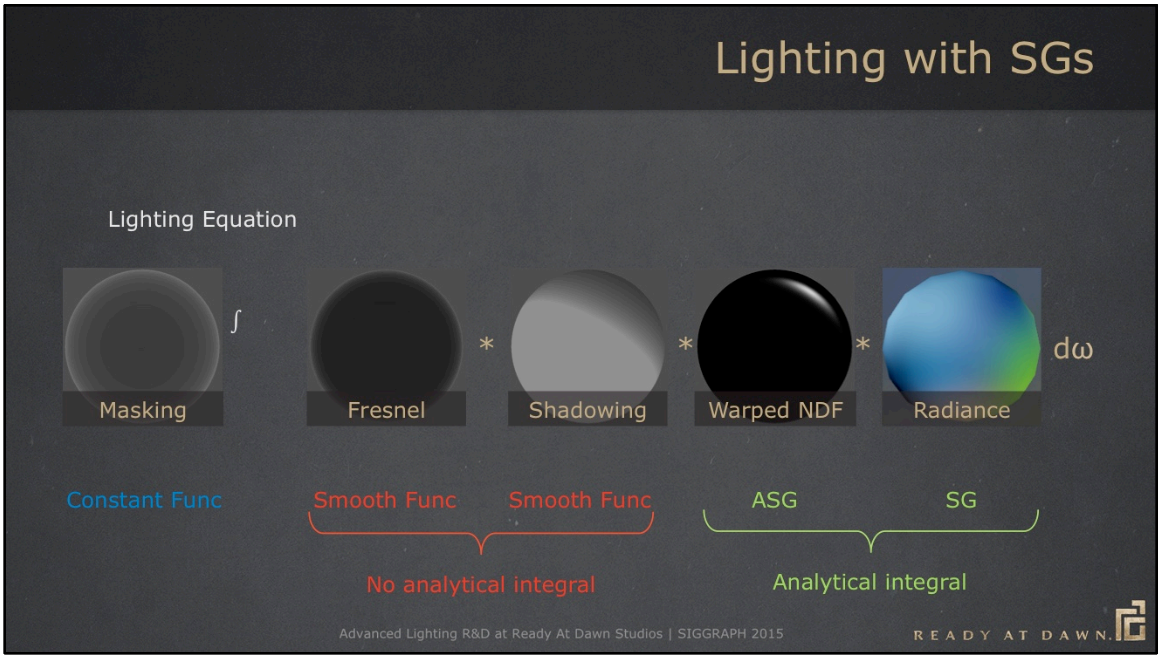
Here is an example of using a symmetric SG warp and an asymmetric ASG warp vs the path traced result.



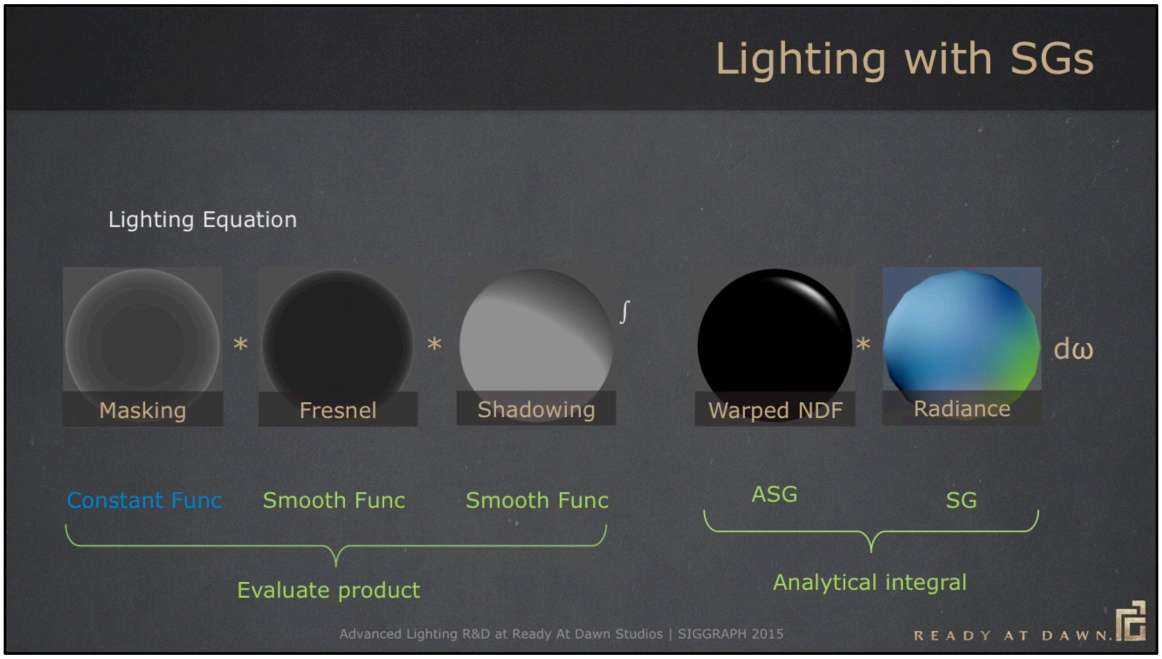
Here is what the lighting equation looks like where I render out the function on the sphere.

The warped NDF is represented as an ASG
The radiance is represented as a sum of SGs
The Fresnel, shadowing and masking functions are the typical functions we use in graphics. Note that the Fresnel function and shadowing function is a fairly smooth function over its domain.

The masking function is actually a constant function with respect to the integral because we are integrating over the domain of the lighting directions while the masking function is parameterized by the view direction.



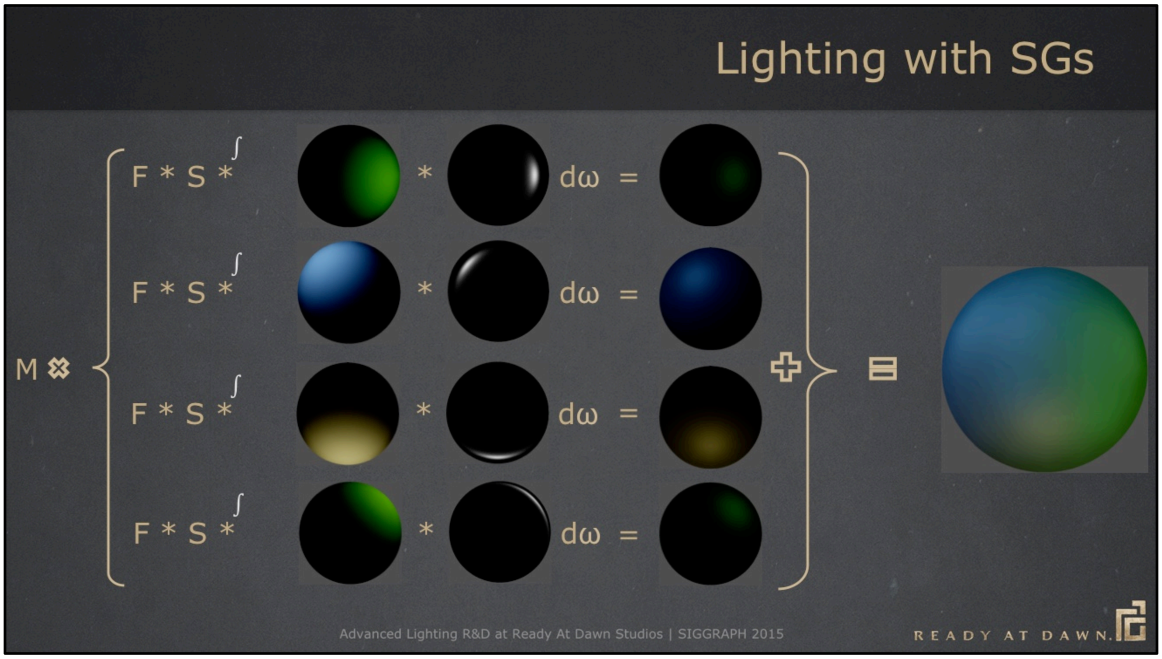
The first thing we can do is pull out the masking function because its constant with respect to the integral.



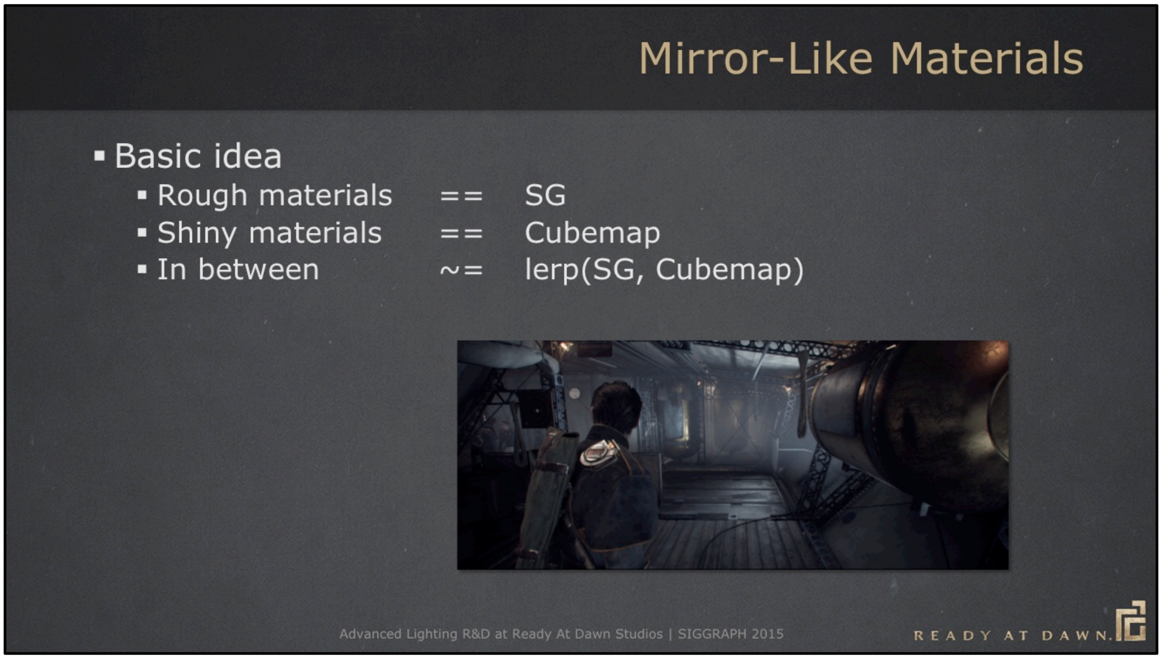
Then since the Fresnel and Shadowing is fairly smooth we can pull it out. This is a fairly strong approximation because the mean value theorem for integration. We need to pick a good representative direction because that will be the average value. We pick the lighting direction to evaluate the Fresnel and shadowing.

This is a fairly strong approximation but if you don't mind paying a little more runtime cost then you can do piece wise linear approximations of the two to get even higher accuracy.

Pulling out the three terms allows us to simply take the product of those function and then multiply it by the analytical integral of an ASG and SG.



To evaluate the final ambient lighting we sum up the integration of the ASG by SG multiplied by their respective Fresnel and Shadowing terms. Then we multiply it by the masking function to get the final result on the right.



In practice

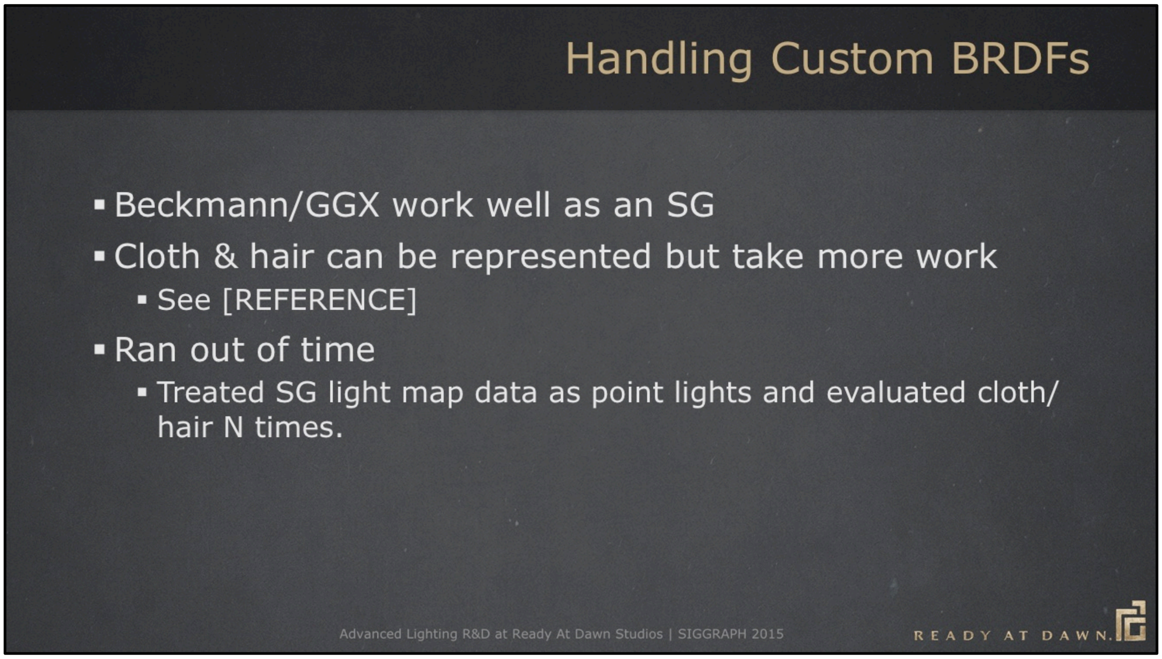
Roughness somewhere ~ .2 in our test scenes

Dependent on # of lobes.

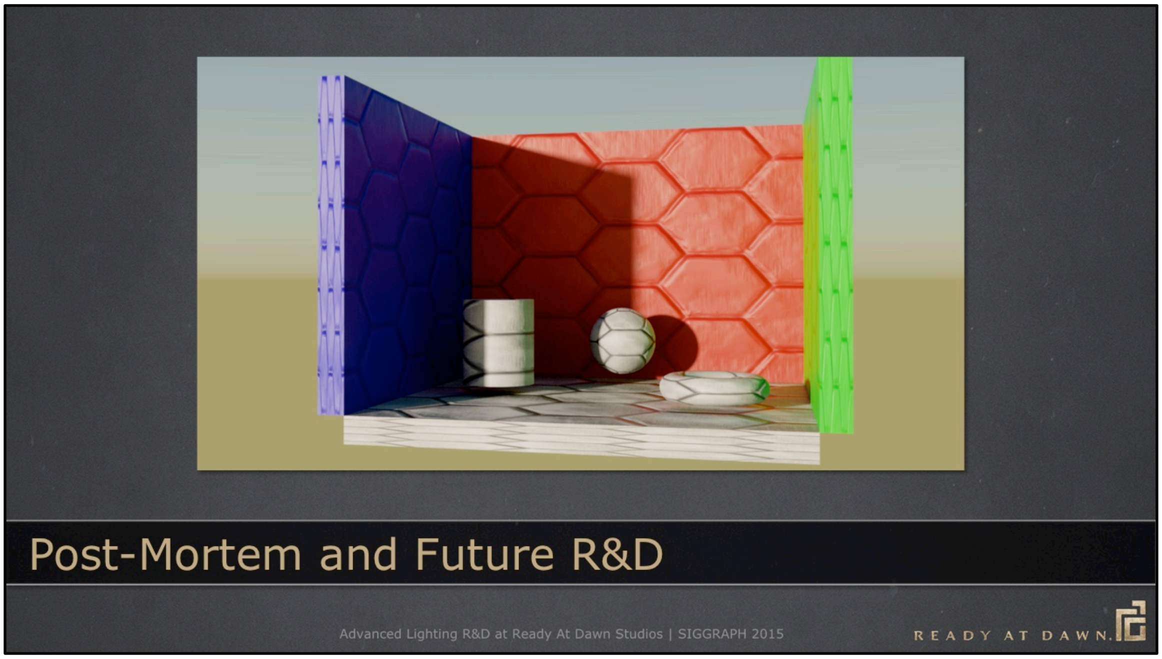
Artists controlled specific blend points in and out per level


Spherical Gaussians / Cube Map Blending Visualization

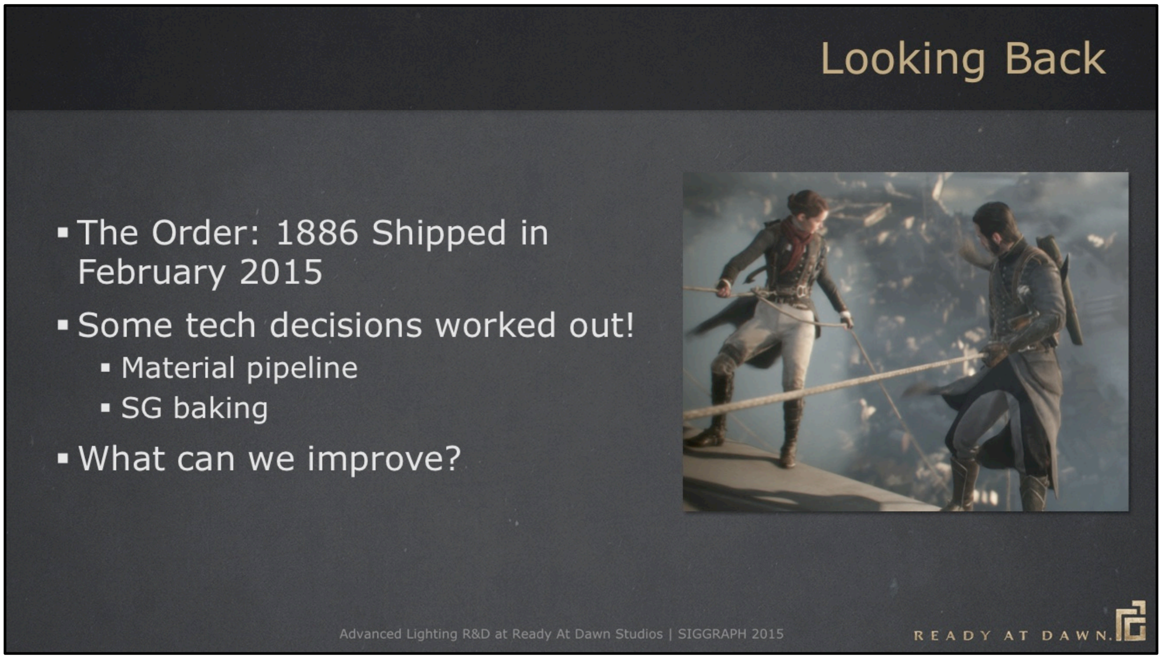
A visualization of a specific choice of blending weights for this scene.



Note: we had fallback for cubemaps with these BRDFs we just didn't like it because the spatial energy problem was worse with cubemaps.



For this section, we're going to take a broader look at the rendering pipeline of The Order: 1886, so that we can evaluate where we could have done better in terms of our technology and authoring pipelines.

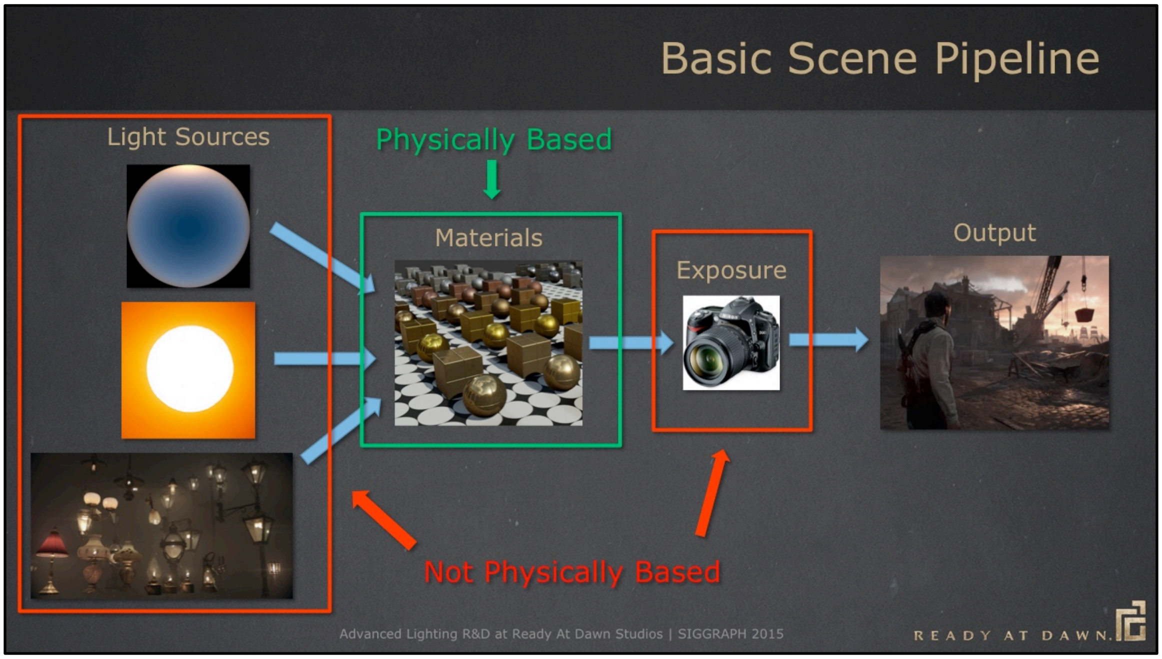


Once we finished production on The Order back in Feburary, it was time for us to take a close look at our technology so that we could evaluate our strengths and weaknesses. In some cases, like the material and shading pipeline that we presented at the 2013 version of this course, it was it was pretty clear that our choices had worked well in terms of creating both good runtime results, as well as an efficient authoring pipeline for our artists. However, there were also plenty of places where we felt that there was plenty of room for improvements that could result in in less time being needed for creating high-quality assets.

When we started the process, we sat down and thought carefully about what our goals were, and what sort of things would be worthwhile candidates for technology improvements. Dave and I both pretty enthusiastic about physically based rendering technology, and sometimes it's easy for us to get excited about changing our tech purely for the sake of having a model that is more closely based on real world behavior. When this happens, I think it's important to consider whether our potential changes are driven by actual, real world needs instead of just our enthusiasm. In that regard, we wanted to make sure that we first started with actual problems encountered during production before proposing solutions.

On a similar note, we wanted to make sure that any potential solutions result in tangible benefits to either the quality or efficiency of artist workflows. This implicitly means that we want to try to avoid new technology that just arbitrarily limits artists in the name of pursuing physically based behavior.

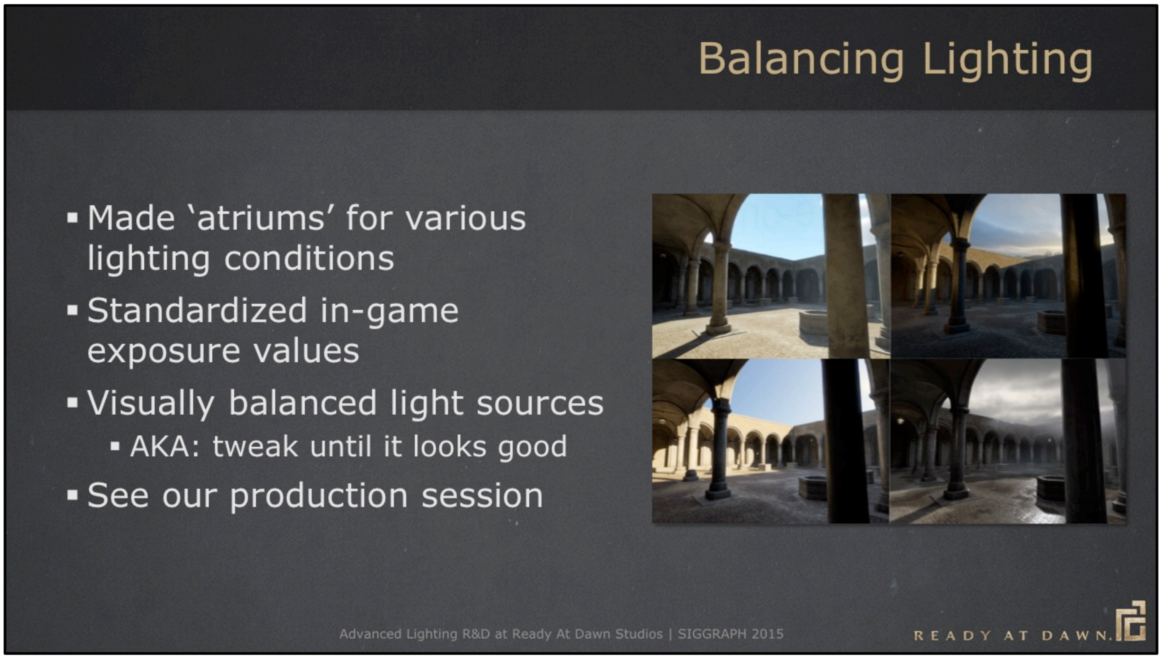
Finally, whenever technology requires artist-driven parameters, we decided that they should always produce physically plausible results by defaults. In some cases it might be necessary to allow artists to break the laws of physics in order to work around inherent limitations and/or approximations, but this should only occur due to explicit



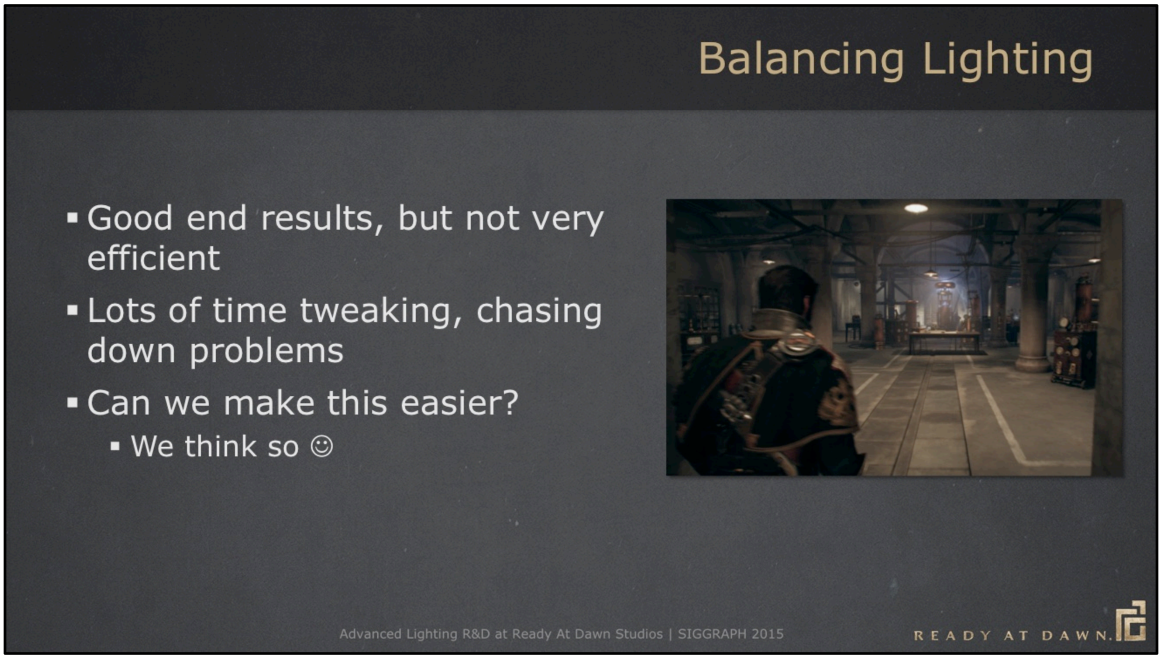
This is a really basic, high-level view of the rendering pipeline that we used for The Order. On the left you have light sources, such as the sky, the sun, and local lighting fixtures. These light sources emit lighting into the scene, which then reflects off the materials in the middle. This reflected lighting goes through an exposure/camera simulation step that converts the incoming per-pixel radiance into the final output in display space.

Out of these steps, only the materials (click) really follow physically based principles. The rest of these steps (click) were not, and used arbitrary units and standards. Somewhat unsurprisingly, these areas in red were the places where we encountered the most problems during production of The Order.

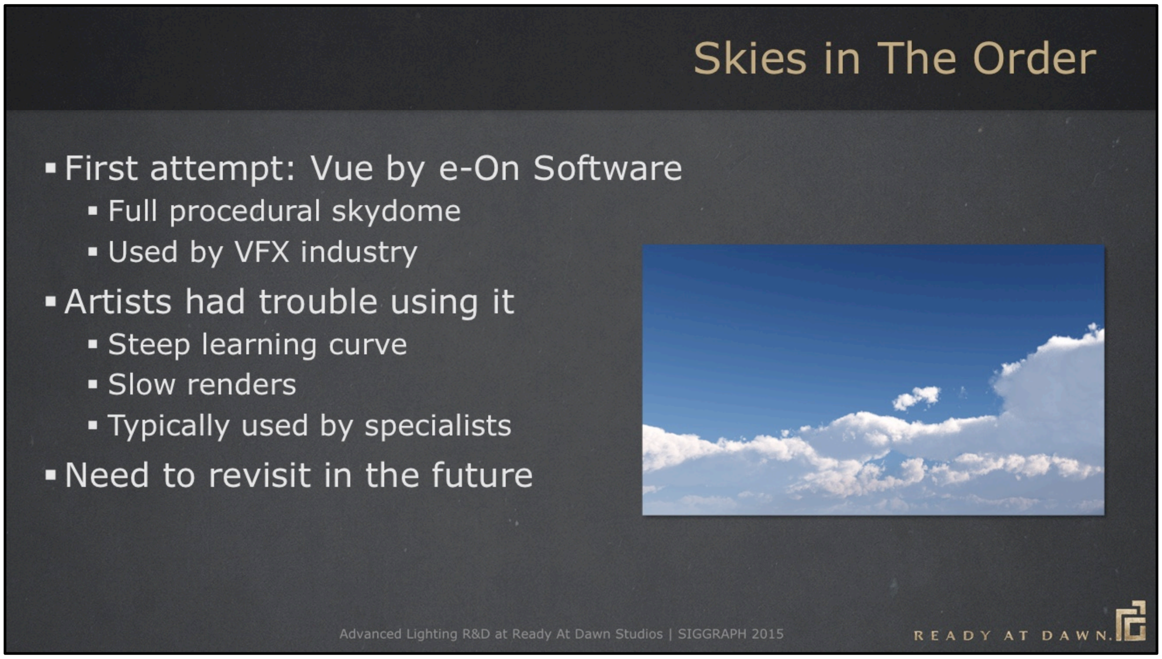
The underlying theme to our problems was balancing various light sources. If you want a realistic scene, it's critical that your various lighting sources are all balanced with one another once they're combined. But for us, the problem was that we didn't really have a fool-proof way of ensuring that this balance was correct when artists were adding light sources.



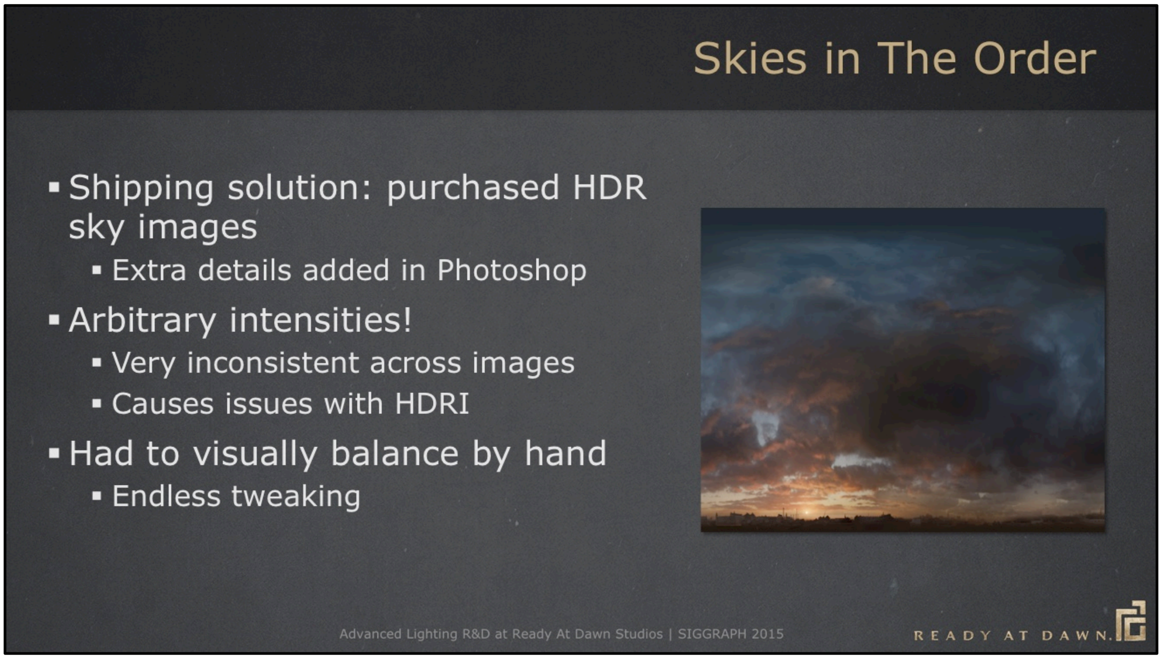
The way we handled balancing was to use test environments for specifying the intensity of lights and various emissive effects. These environments all used the same atrium geometry, which was coupled with various sun and sky components that reflected varying weather conditions as well as times of day. The artists chose standardized exposure values in these atriums, which were typically used as base exposure values for similar in-game conditions. To balance the light sources, they were simply added to the scene and their intensity was tweaked until they appears to be visually coherent with the rest of the scene. This process was somewhat error-prone, and time consuming as well.



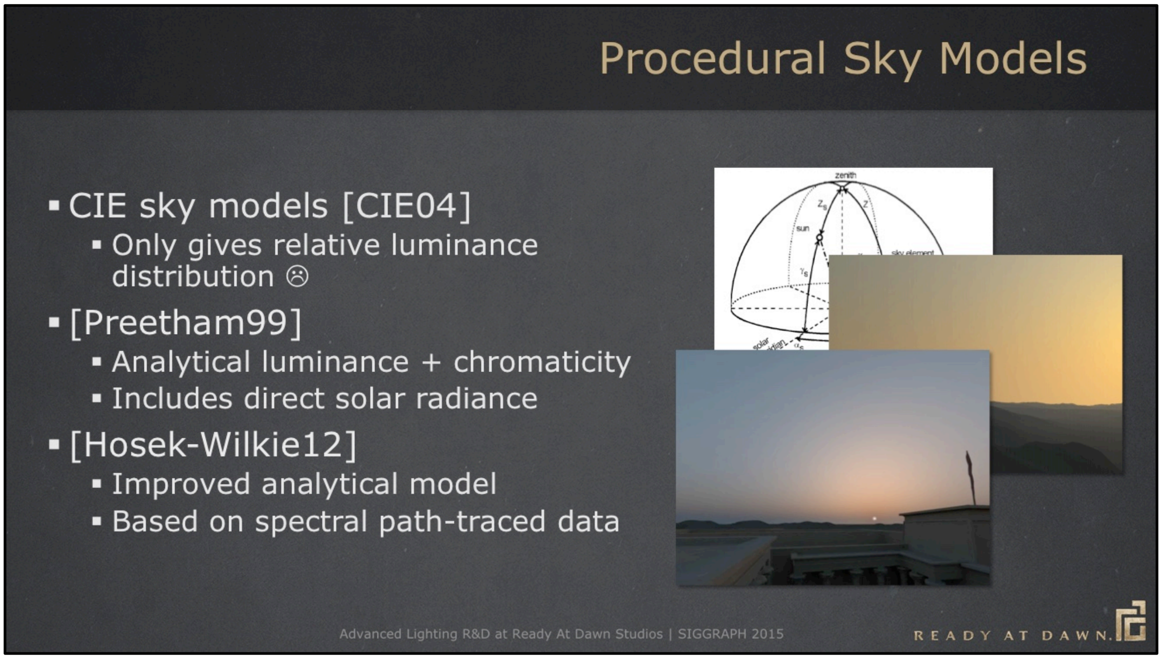
The takeaway that we had after the project was that while it was definitely possible to create compelling visuals with our process, it definitely was as efficient as it could be. We firmly believe that applying more physically based principles to our pipeline could help us to end up with a more streamlined workflow.



Early in the project, we evaluated Vue by e-On Software as a tool for authoring HDR skydomes. It's a full procedural solution that's very powerful, but it also has a pretty tough learning curve. At VFX studios that use the tool, it's common to have a dedicated Vue artist that's an expert in all the parameters exposed by their simulation. We had no such expert, and so the artists decided to pursue a more low-tech solution.



Most of the skies used in the final game were based on HDR images that were purchased from various third-party libraries. Our vista artists found these much easier to work with, since they could use familiar tools like Photoshop and Mari to customize the image with additional details. The major issue with this approach was that the HDR image were all over the place in terms of their intensity. They didn't utilize any particular physical units for the pixel values, and so we had to manually tweak the overall intensity until they fit with the arbitrary units used by our game.

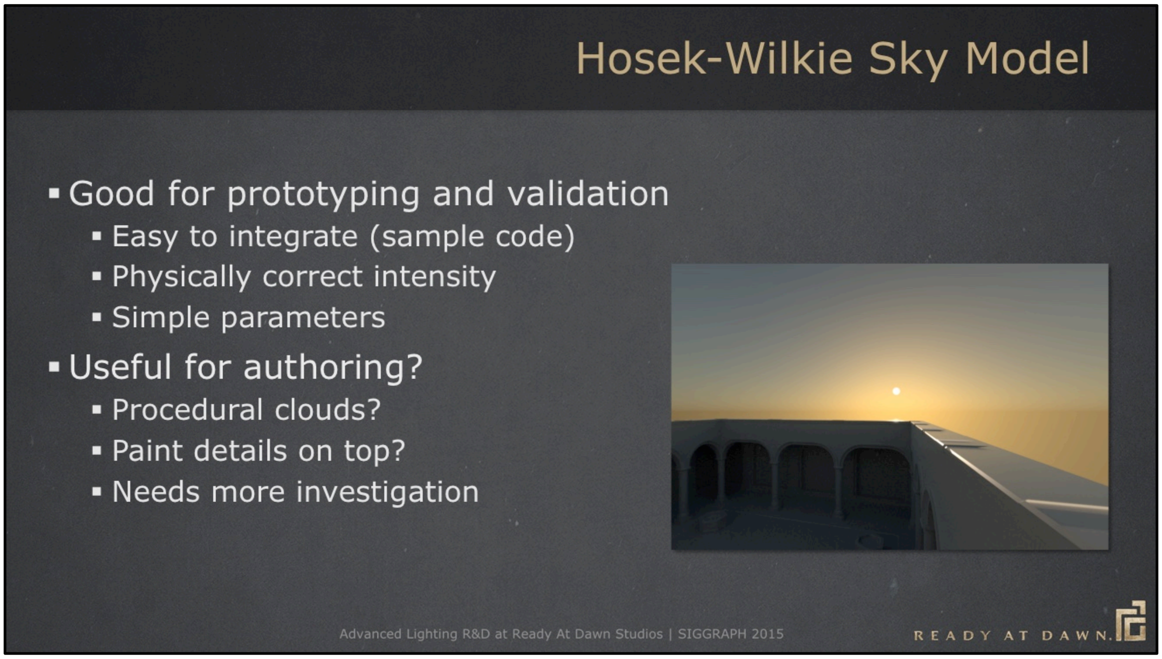


As part of our R&D, we've started investigating various procedural sky models. There are a few such models that are popular in graphics research.

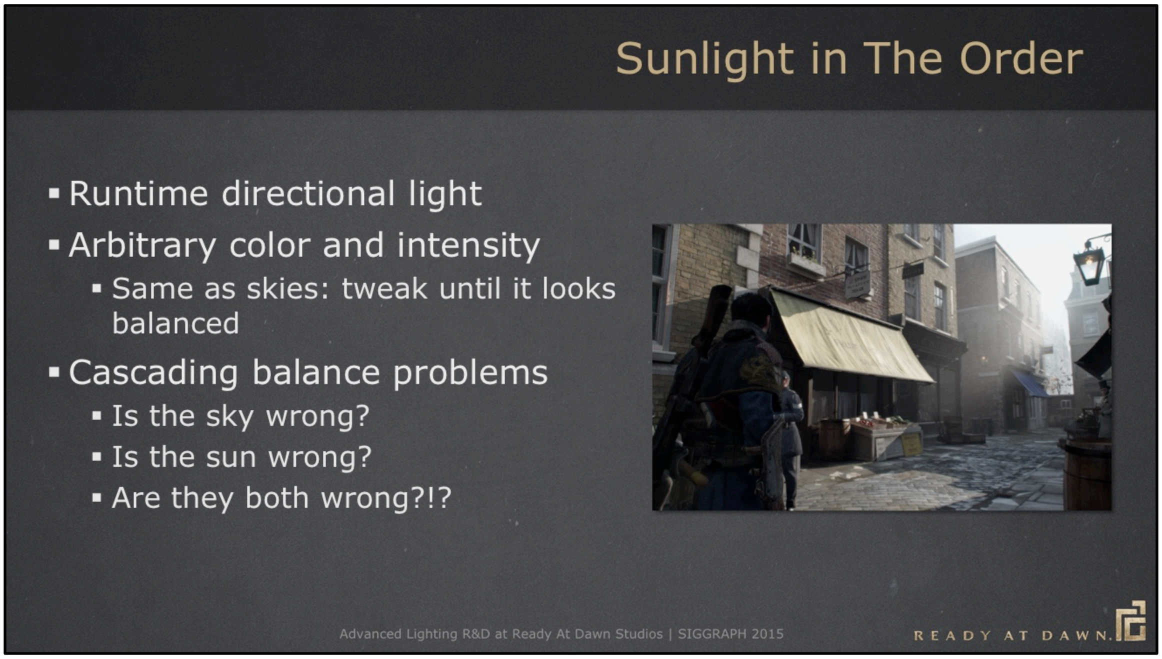
The CIE general sky is primarily designed for architectural models, and doesn't give you absolute luminance values. It also has no chromaticity whatsoever. This makes it pretty much unsuitable for our purposes, since at the very least we'd like to have a physically correct intensity.

The Preetham model is pretty old at this point, but has gotten a lot of use. It provides an analytical means of computing the luminance and chromaticity for any given point on the sky, and has only a small set of parameters.

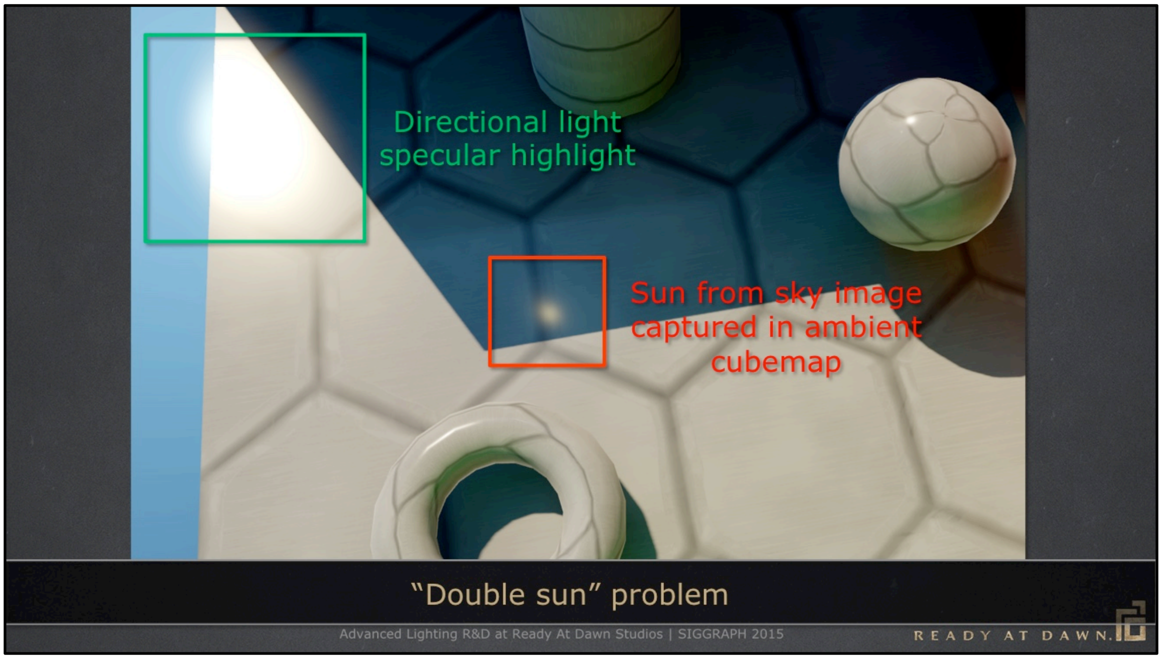
The Hosek-Wilkie model is fairly new, and aims to improve on a few deficiencies present in the Preetham model. Like Preetham, it provides full RGB data with a similarly-small set of parameters.



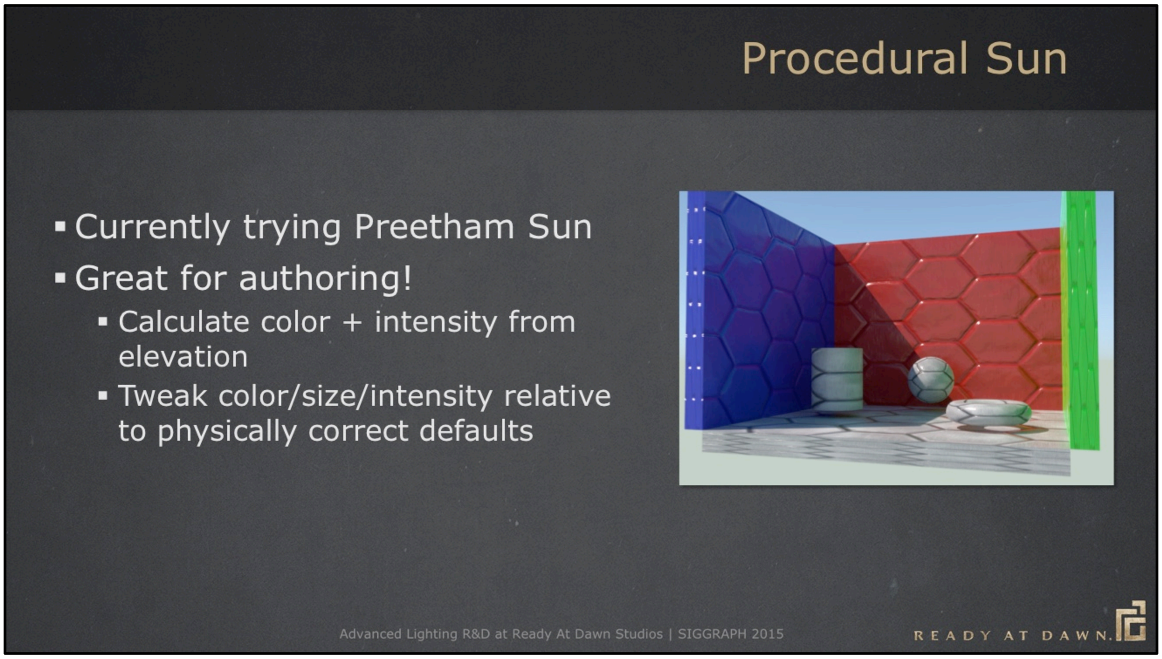
We're still in the early stages of R&D, but so far the Hosek-Wilkie model has proven to at least be useful for prototyping and validation. The authors provide a sample implementation as C code, and it was pretty trivial for us to integrate that into our testbed application. Once integrated we were immediately able to get balanced renders that had the appropriate lighting intensity. We've also had some success at using it in actual game scenarios, which is done by generating EXR images that are imported into the game engine. However, it is still unknown whether or not it will be useful as a means of authoring the final skies used in a shipping product. To know this, we need to spend time investigating the means by which we might add additional details to the sky, such as clouds mountain ranges.



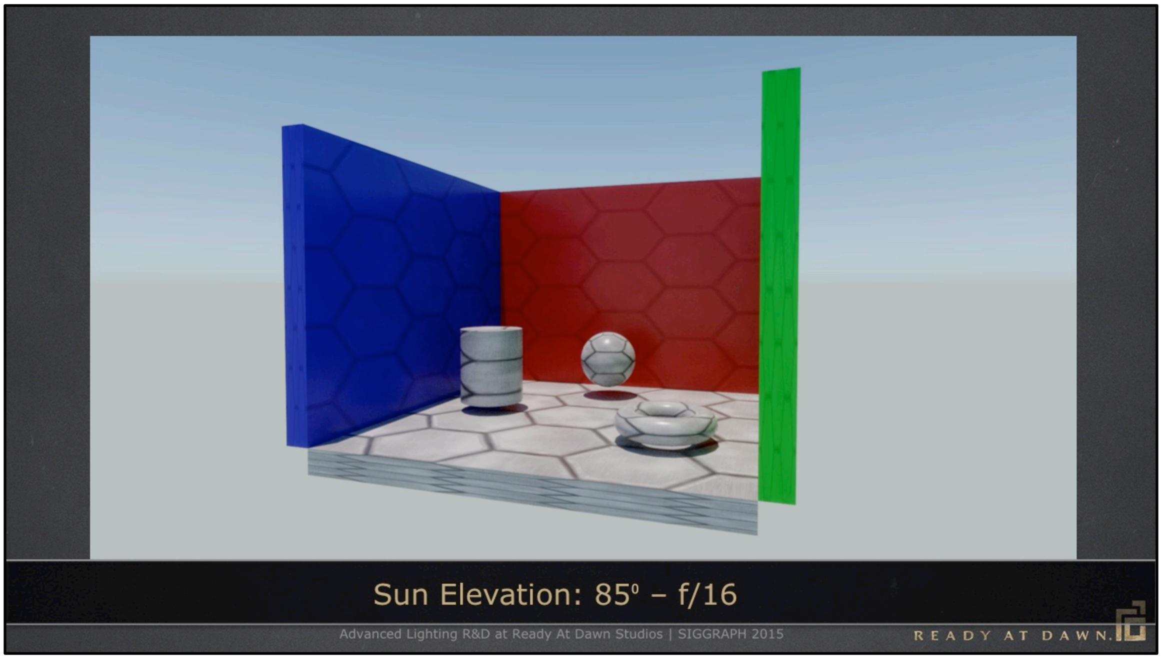
For modeling illumination for the sun in The Order, we utilized a runtime directional light with hand-picked intensity values. Like the sky, this light source used arbitrary units, and thus it was up to the artists to visually verify that the sunlight looked balanced with the sky. The problem that arose was that we now had an additional unknown to deal with when trying to debug issues that cropped up during production. For example, if a particular specular highlight from the sun appeared too bright, we didn't know which component was causing the problem. Was it that the sun was too bright? Or perhaps the sky was just too dark. Or perhaps the material was wrong. Without a real-world frame of reference for lighting units, we were often left scratching our heads at these sorts of problems.



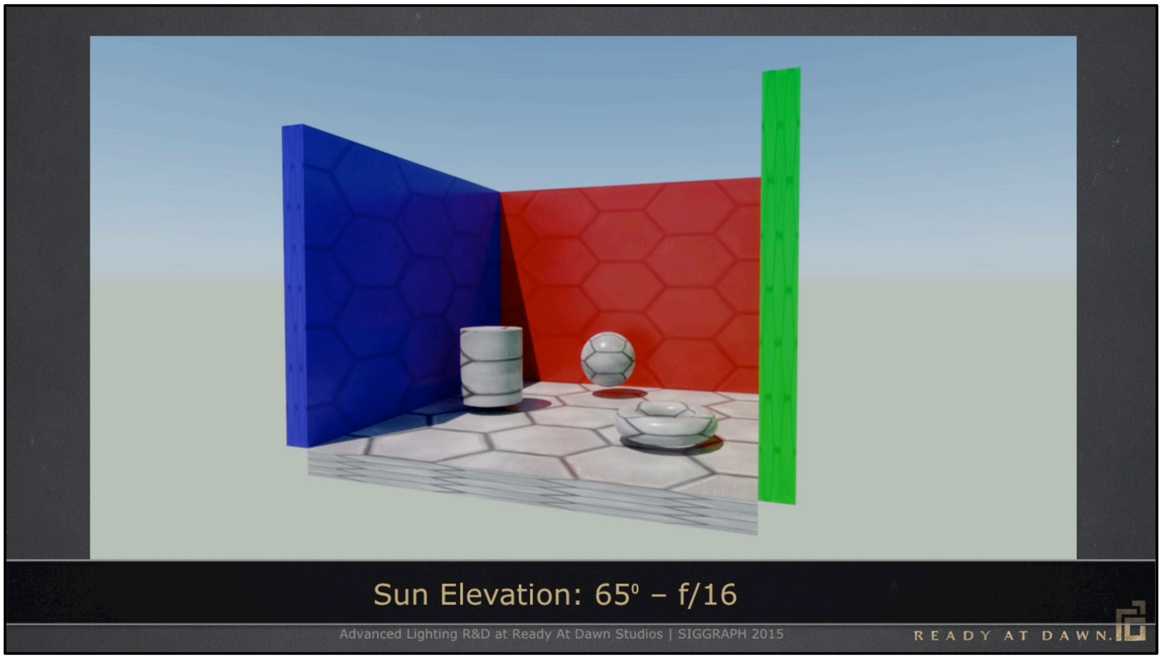
This image shows another common issue with sunlight that you can run into if you're using captured cubemaps as IBL specular probes. If your sky image has a solar disc, and you capture that in your cubemap probe, you'll see that solar disc show up in the runtime reflections. Often the resulting highlight will be too dark due to incorrect intensity, and it will show up in shadows due to poor spatial locality of the specular probes. With a full procedural model it is trivial to handle this: just leave out the solar disc when capturing your specular probes. However if it's in the sky image, then you have to find a way to remove it. For The Order, we would often try to paint over the sun, or block it with occluder geometry.

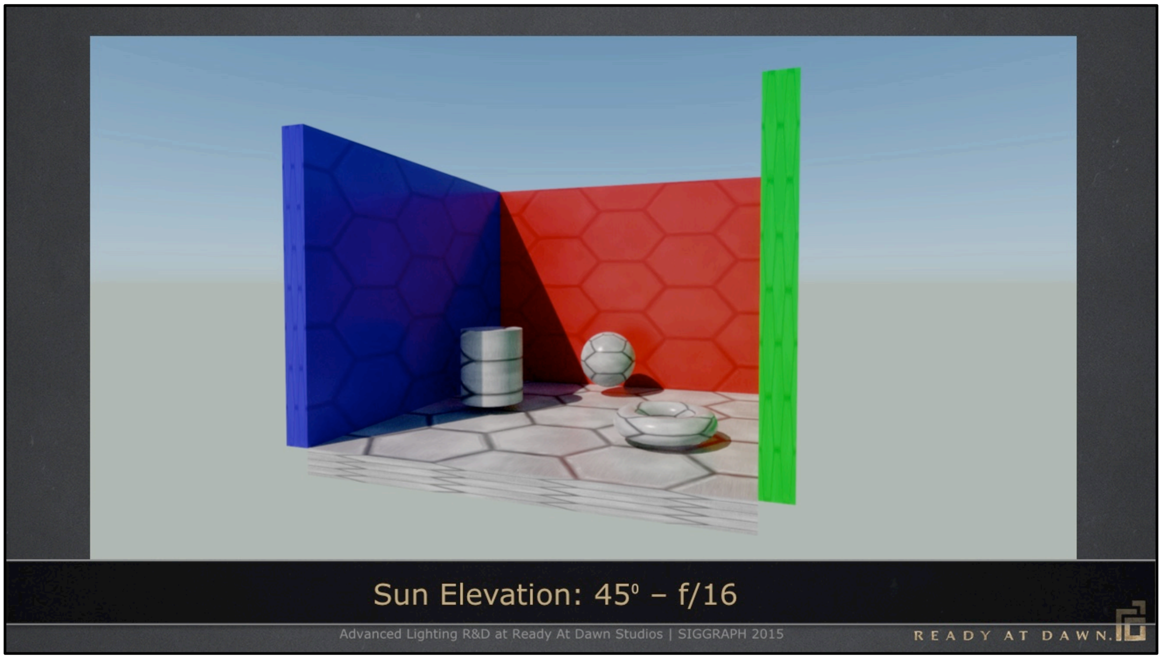


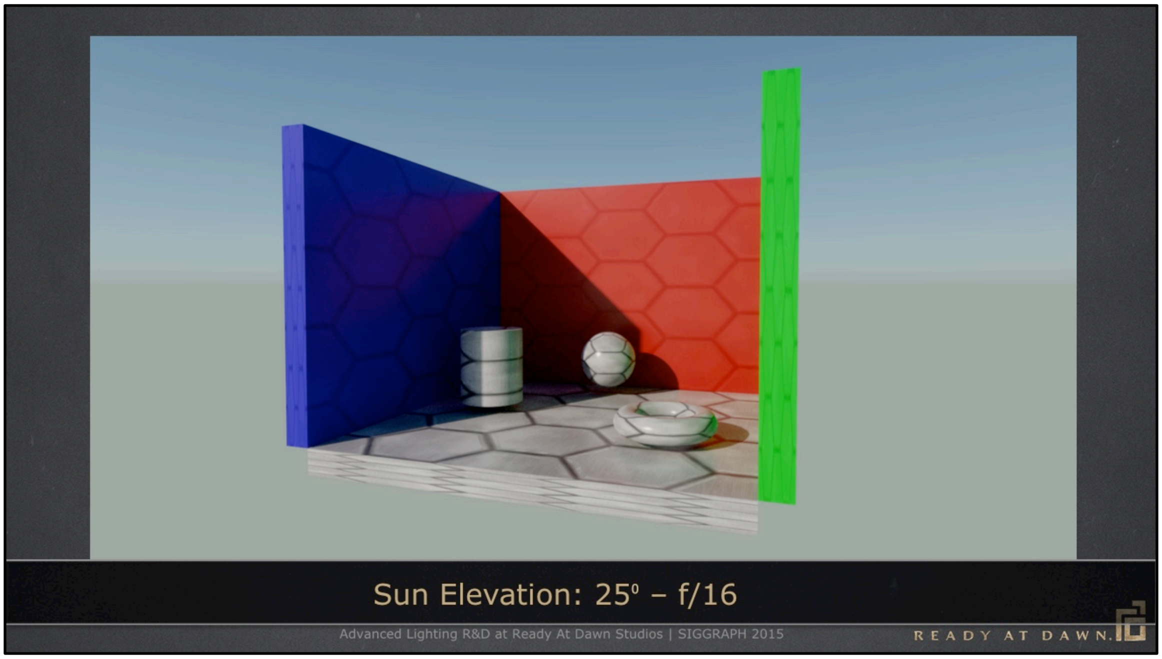
To help resolve some of these issues, we've also begun experimenting with using a procedural sunlight model for illuminating the scene. Currently we've been using the Preetham model for computing solar radiance, but the Hosek-Wilkie model also provides a solar radiance function that includes simulation of limbal darkening. We've had great success with using it as an authoring tool. All you need is the sun's current elevation, and you can compute the appropriate color and intensity for the solar disc. With some simple math, it's also possible to compute the resulting irradiance given the size of the sun, which can then be used to determine an approximate intensity for a directional light.

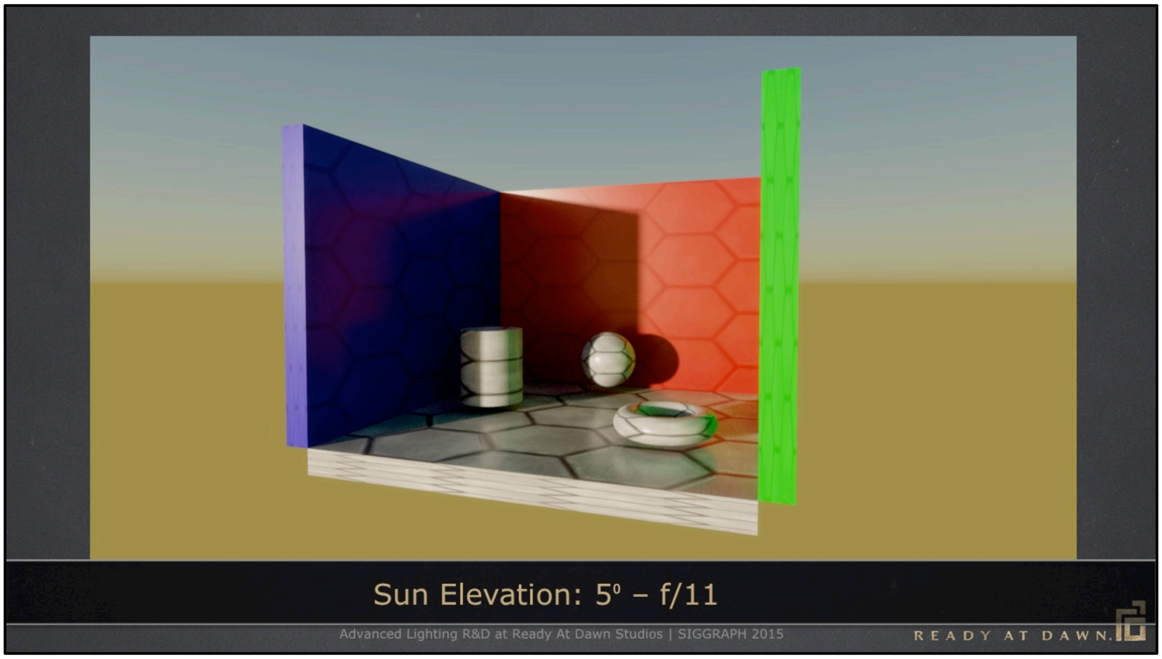


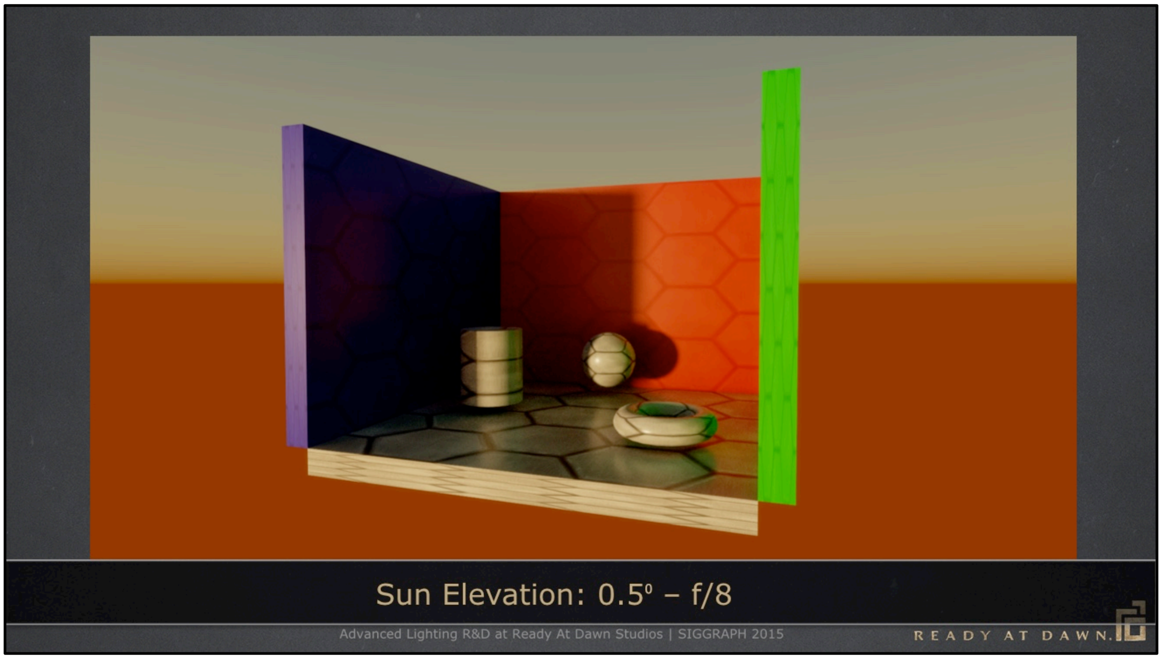
This a set of images rendered in our custom path tracer, using the procedural sun and sky model. These aren't meant to be impressive, they just how easy it was to render balanced lighting with appropriate intensities. You can see how the aperture size changes as the sun goes down in order to match the corresponding drop in sun intensity.


Sun Elevation: 65⁰ – f/16
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015
READY AT DAWN.


Sun Elevation: 45⁰ – f/16
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015
READY AT DAWN.

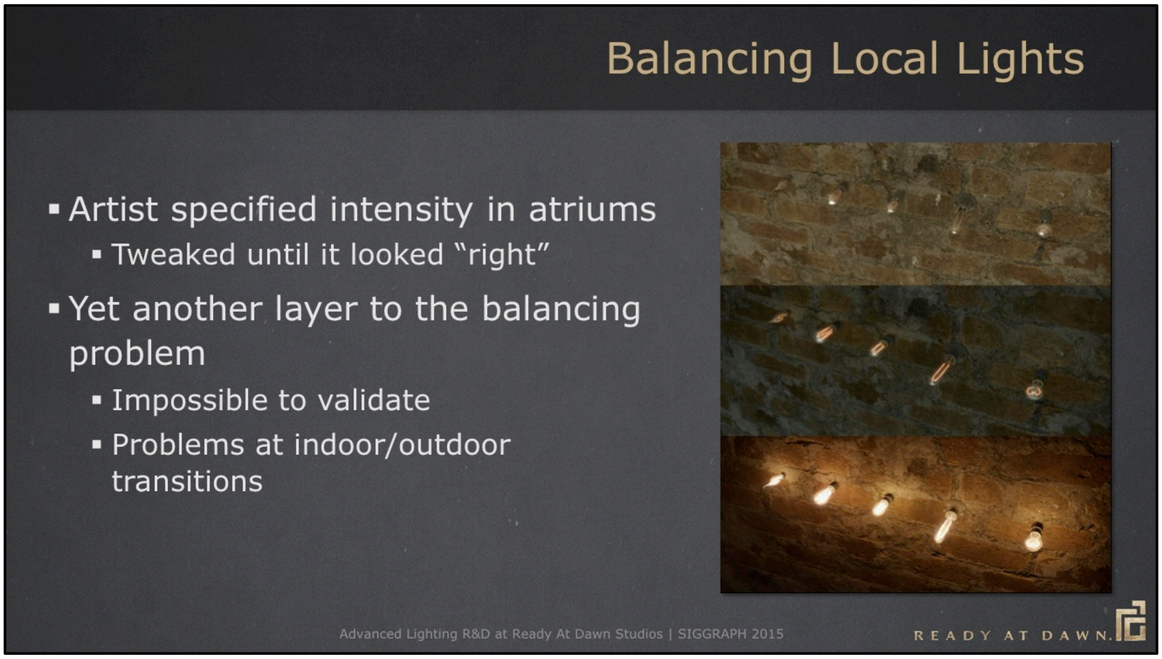

Sun Elevation: 25⁰ – f/16
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015
READY AT DAWN.


Sun Elevation: 5⁰ – f/11
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015
READY AT DAWN.


Sun Elevation: 0.5° – f/8
Advanced Lighting R&D at Ready At Dawn Studios | SIGGRAPH 2015
READY AT DAWN.

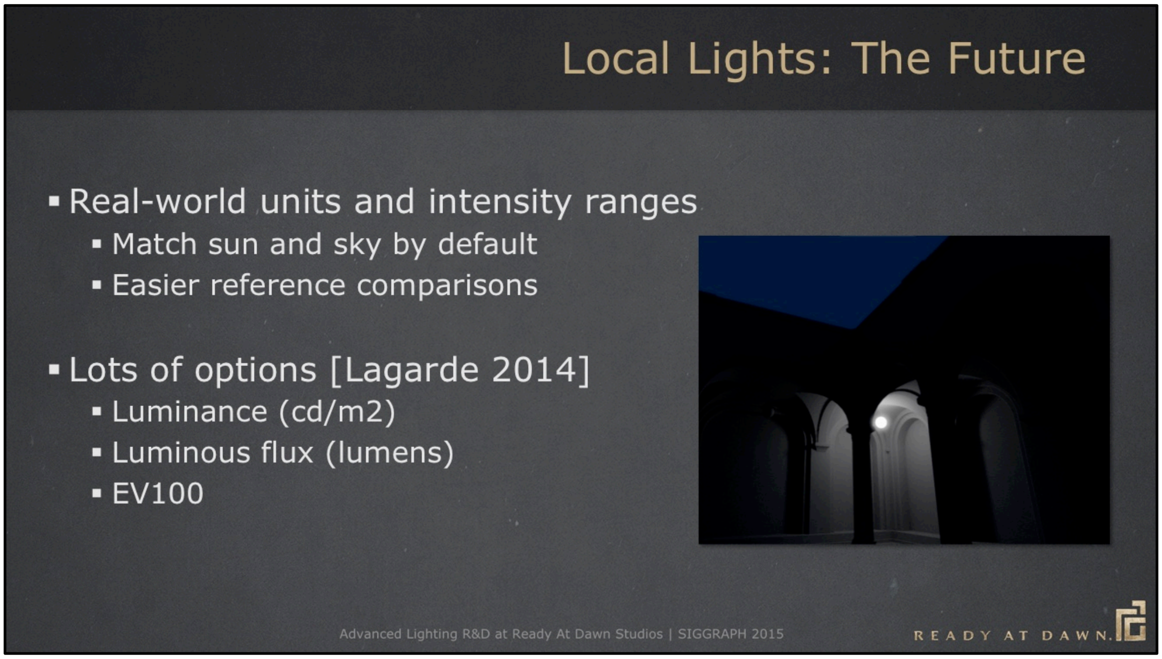
The Order featured a variety of local light sources, which were primarily modeled as simple point and spot lights at runtime. Like the sun and sky, they didn't use physical units and instead used arbitrary intensity values.

One thing that we quickly realized was that we weren't going to be able to individually choose parameters for every single light source in the game. To cope with this, we created a database of light fixtures that could be quickly placed into a scene using Maya. These fixtures contained both the runtime light source, as well as the geometry representation for the light and its surrounding fixture.



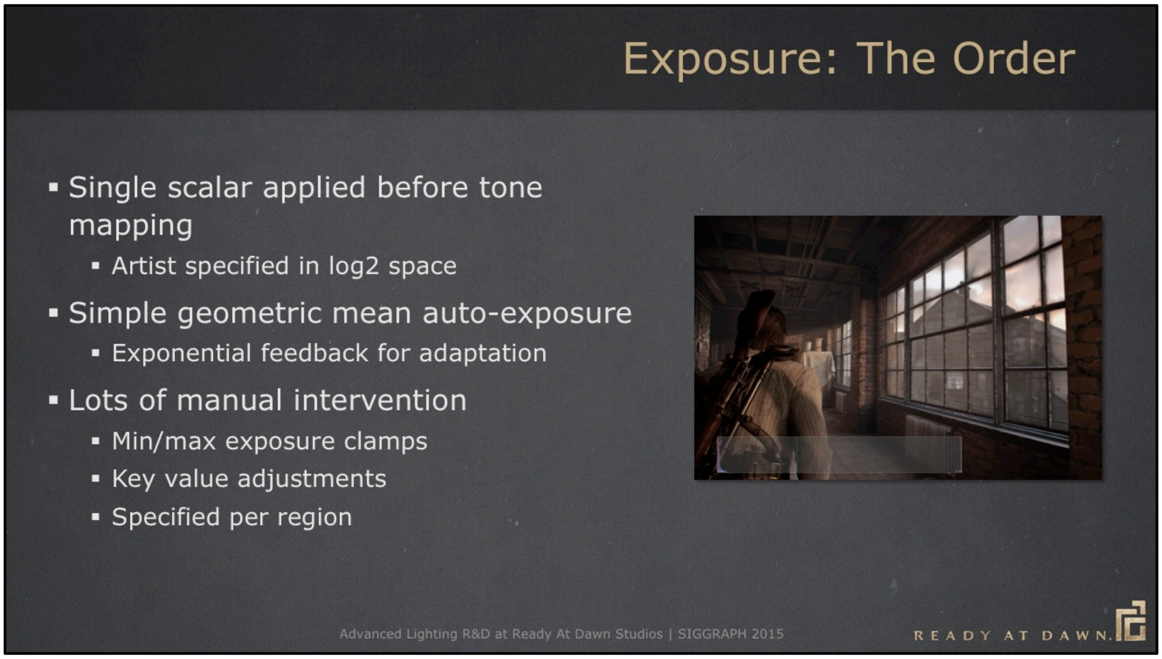
Since we didn't use physical units, the lighting artists needed to manually verify that the resulting lighting looked balanced when combined with the sun and sky. To do this, they again used the test atriums to visually inspect both the light contribution as well as the emissive component. Basically, the artists would just drop the lights into a certain lighting condition, and then try to make sure that the light source looked the way that they would expect it to work.

The major problem here was that it was basically impossible to validate anything once local lights were in the mix. It was really just another layer on to of the balancing problem, which added yet another possible source for unbalanced lighting. We also had some issues in transition areas between two lighting conditions. In some cases things were set up well for one particular condition, but when the exposure changed to a value appropriate for a new lighting condition the balance would appear incorrect.



In the future, we think that using physical units for light sources could be a big help. It's easy to look up actual values for various light fixtures, and so those could be used in-game to make sure that the resulting illuminance is something that the artist expects. It could also be extremely helpful for matching our engine to real-world reference scenes.

Sebastian Lagarde presented a few options last year for using physical units, and we're still doing R&D to decide what will work best for us.

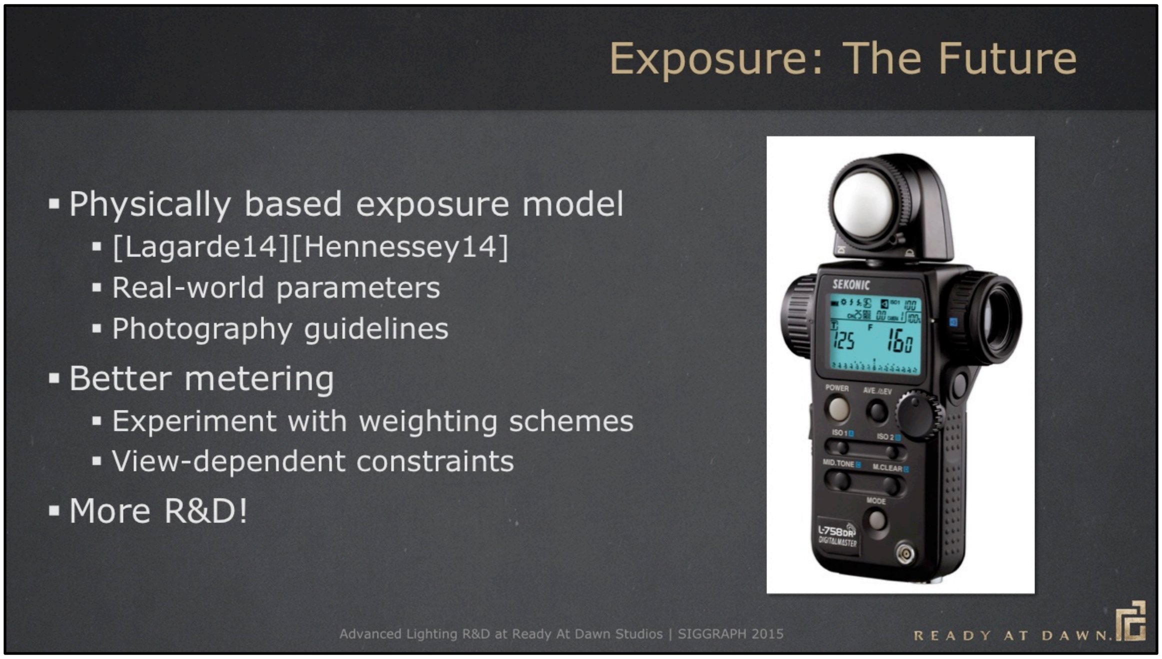


Exposure is another component of the overall problem, since it's responsible for taking the reflected per-pixel lighting values and scaling them into a range that's suitable for tone mapping into display space. In The Order, our exposure was just a simple scalar value that was directly applied to each pixel right before the tone mapping stage. Consequently, there wasn't any particular mapping to real camera parameters or exposure standards. We did allow the artists to specify it in Log2 space, in order to at least somewhat match the response of parameters used in real-world cameras.

For auto-exposure, we took the fairly common approach of computing the geometric mean of luminance for the entire screen, and then picking a scale value that would map that average luminance to a particular middle grey value. As lots of other studios have found out, this technique can have a lot of problems. Since its based on taking an equal average luminance of the screen, it can often end up choosing an exposure that's not well-exposed for any particular element of the scene. We ended having to use a lot of manual intervention to get acceptable results, which mostly took the form of the min and max clamp values that were applied in regional volumes. This is obviously not the kind of workflow that we want for a system that's supposed to be automatic.

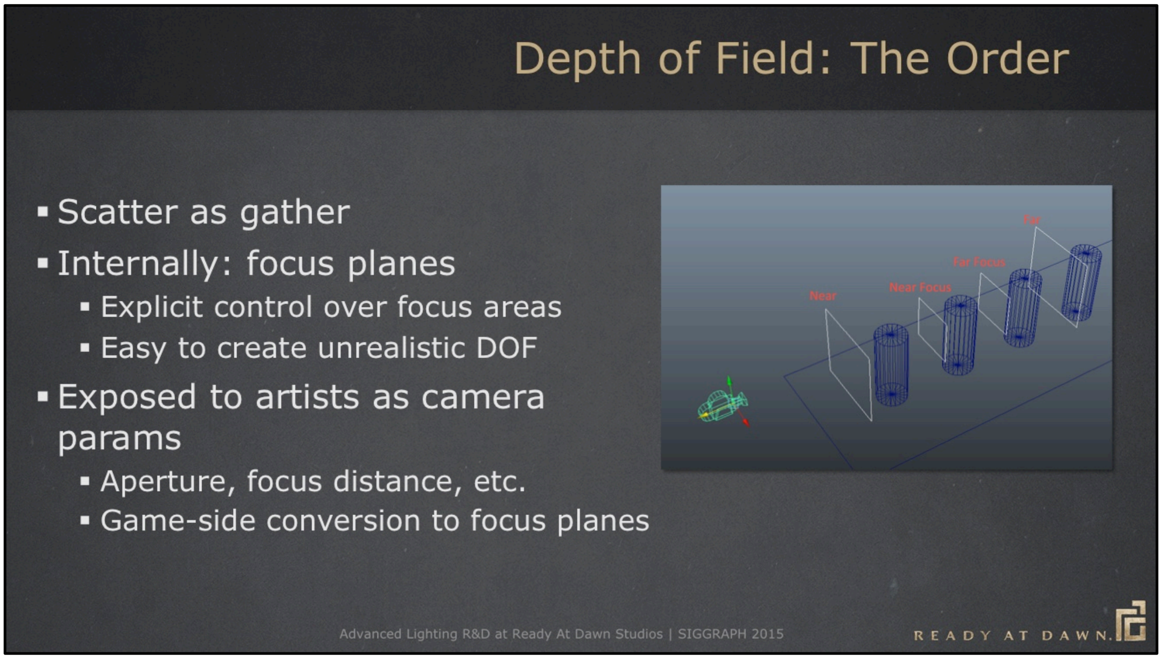
To show an example of how auto-exposure typically failed for us, here's an in-game capture taken from the Blackwall Yards level. In this screenshot, automatic exposure is enabled with default settings. The resulting exposure doesn't really work, since the vista seen through the window is too blown out, washing out all of the details. This happens because the system is trying to compensate for the darker areas on the bricks, which are darker both because of the interior lighting as well as because of the darker albedo.

Here's the same screenshot, but with an artist-specified exposure value instead of using the automatic exposure system. The result is much more balanced than the previous shot, since you can see all of the key details and silhouettes in the vista area.



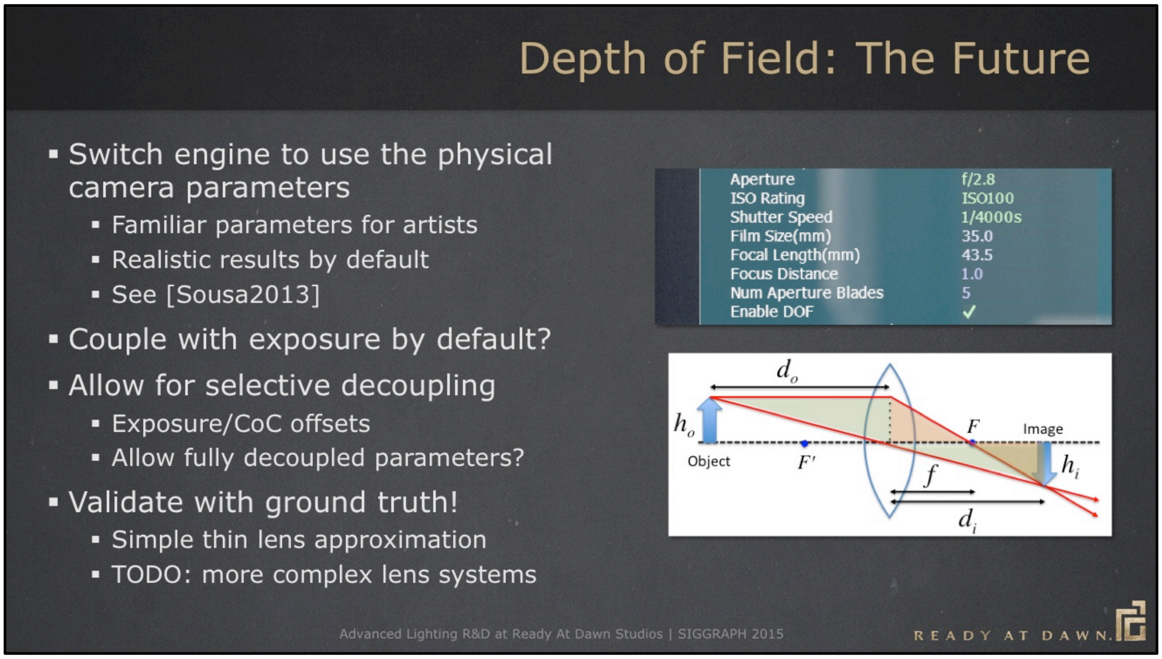
For future projects, we're planning on adopting a more coherent physically based exposure model. We've been experimenting with driving exposure with actual camera parameters such as aperture and EV's, and the results have been promising. Using real parameters makes the exposure more consistent with physically based lighting intensities, and allows us to use common photography conventions for choosing exposure.

For automatic exposure, we are currently looking into alternative techniques for metering and weighting schemes. I firmly believe that we can do a better job of metering by making more use of information that's inherently available to a game. For instance, in our games we typically have a player character right in front of the camera, and so it should be possible to weight the player pixels higher so that the model always ends up well exposed. There have also been recent games that take the approach of computing exposure based on the incident irradiance of surfaces rather than using the final reflectance. Doing this means that albedo and specular don't affect the result, which is exactly what you want in a lot of cases.



This is somewhat unrelated to the other topics that I just discussed, but I bring it up since it's another example of where we're moving to physically based parameters. At an engine level, our depth of field was driven by 4 focus planes. Basically we would use the pixel depth to blend up to a certain CoC size based on where that depth value ended up relative to the planes. If these parameters were used directly, they basically gave the artist direct control over which areas were in-focus and which areas were out-of-focus, which didn't have to be grounded in any real-world camera behavior. As a result, it was really easy to create a totally unrealistic depth of field effect, and rather hard to choose settings that looked realistic. At various points in the project, I brought up the idea of using real camera parameters to our artists and designers, so that they could get realistic results by default. However at the time, we felt that having absolute control was more desirable.

As production went on, some of our gameplay camera systems adopted few camera parameters to the artists, and then performing an ad-hoc conversion to focus planes. These camera parameters ended up working out very well, disproving our earlier concerns about loss of control.



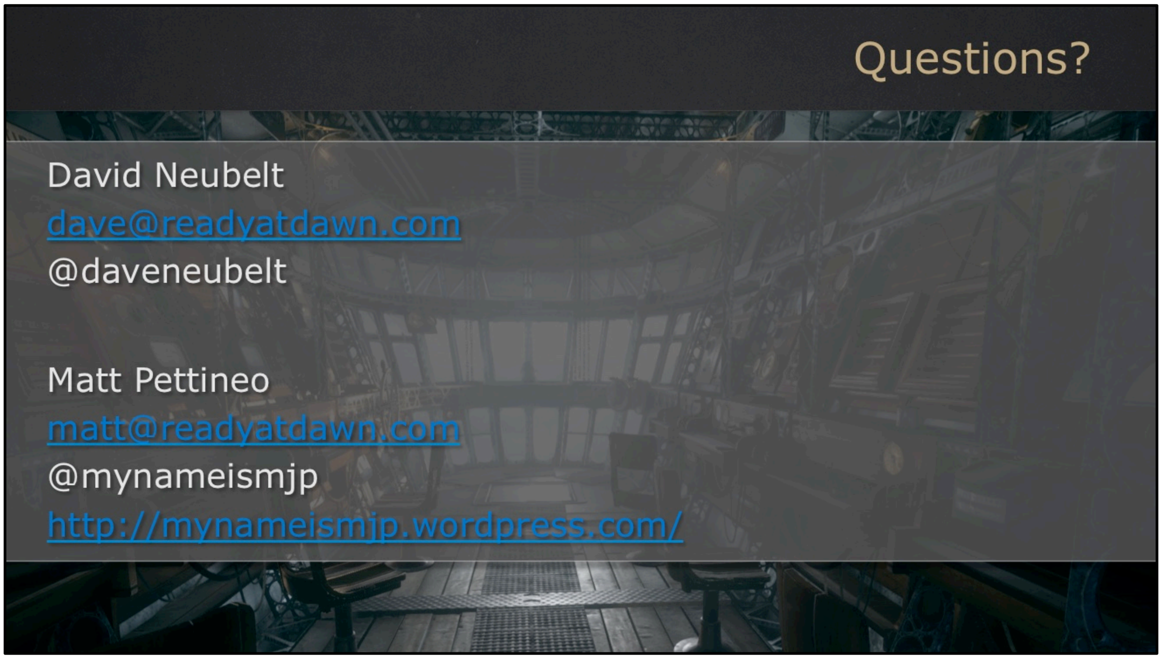
Based on our past experiences, it's now clear to us that real parameters should be used throughout the whole pipeline. One thing that is somewhat less clear is whether or not we should couple the depth of field parameters to exposure, in order to mimic the behavior of a real camera. At the moment, we're leaning towards coupling them but with selective controls for biasing the behavior of either system for extreme cases. However it will take a lot more real world experience before if we know whether or not that's a good decision.

One more thing I'll mention is that we've started validating our depth of field approximation against a ground truth path tracer. Doing this helps spot bugs and other issues that cause the real-time approximation to look different from ground truth. One thing I still need to do for this is improve the ground truth simulation with more complex lens systems, such as the ones you find in DLSR cameras. Doing this might help to inform the real time version, so that we can get higher quality results.

Acknowledgements

- Stephen Hill
- Stephen McAuley
- Nick Blasingame
- Joe Schutte

.


Questions?
David Neubelt
dave@readyatdawn.com
@daveneubelt
Matt Pettineo
matt@readyatdawn.com
@mynameismjp
http://mynameismjp.wordpress.com/

.

References

[CIE04] – Spatial Distribution of Daylight – CIE Standard General Sky
[Preetham99] – A Practical Analytic Model for Daylight
[Hosek12] – An Analytic Model for Full Spectral Skydome Radiance
[Hosek13] – Adding a Solar-Radiance Function to the Hosek-Wilkie Skylight Model
[Lagarde14] – Moving Frostbite to PBR
[Hennessey14] – Implementing a Physically Based Camera – Manual Exposure
[Sousa13] - Graphics Gems from CryEngine 3
[Wang09] – All-Frequency Rendering of Dynamic, Spatially-Varying Reflectance
[Xu13] – Anisotropic Spherical Gaussians
[Xu14] – A Practical Algorithm for Rendering Interreflections with All-Frequency BRDFs

.