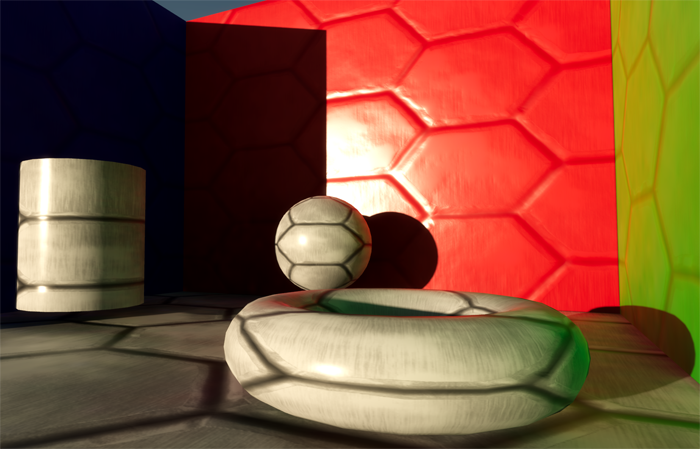
Oct 10, 2016

**SG Series Part 6: Step Into The Baking Lab**



*This is part 6 of a series on Spherical Gaussians and their applications for pre-computed lighting. You can find the other articles here:*

Part 1 - [A Brief (and Incomplete) History of Baked Lighting Representations](https://therealmjp.github.io/posts/sg-series-part-1-a-brief-and-incomplete-history-of-baked-lighting-representations/)  
Part 2 - [Spherical Gaussians 101](https://therealmjp.github.io/posts/sg-series-part-2-spherical-gaussians-101/)  
Part 3 - [Diffuse Lighting From an SG Light Source](https://therealmjp.github.io/posts/sg-series-part-3-diffuse-lighting-from-an-sg-light-source/)  
Part 4 - [Specular Lighting From an SG Light Source](https://therealmjp.github.io/posts/sg-series-part-4-specular-lighting-from-an-sg-light-source/)  
Part 5 - [Approximating Radiance and Irradiance With SG’s](https://therealmjp.github.io/posts/sg-series-part-5-approximating-radiance-and-irradiance-with-sgs/)  
Part 6 - [Step Into The Baking Lab](https://therealmjp.github.io/posts/sg-series-part-6-step-into-the-baking-lab/)

Get the code on GitHub: <https://github.com/TheRealMJP/BakingLab> (pre-compiled binaries available [here](https://github.com/TheRealMJP/BakingLab/releases))

Back in early 2014, myself and David Neubelt started doing serious research into using Spherical Gaussians as a compact representation for our pre-computed lighting probes. One of the first things I did back then was to create a testbed application that we could use to compare various lightmap representations (SH, H-basis, SG, etc.) and quickly experiment with new ideas. As part of that application I implemented my first path tracer, which was directly integrated into the app for A/B comparisons. This turned out to be extremely useful, since having quick feedback was really helpful for evaluating quality and also for finding and fixing bugs. Eventually we used this app to finalize the exact approach that we would use when integrating SG’s into The Order: 1886.

A year later in 2015, Dave and I created another test application for experimenting with improvements that we were planning for future projects. This included things like a physically based exposure model utilizing real-world camera parameters, using the [ACES](https://en.wikipedia.org/wiki/Academy_Color_Encoding_System)[1] RRT/ODT for tone mapping, and [using real-world units](http://www.frostbite.com/wp-content/uploads/2014/11/course_notes_moving_frostbite_to_pbr_v2.pdf)[2] for specifying lighting intensities. At some point I integrated an improved version of SG baking into this app that would progressively compute results in the background while the app remained responsive, allowing for quick “preview-quality” feedback after adjusting the lighting parameters. Once we started working on our [SIGGRAPH presentation](http://blog.selfshadow.com/publications/s2015-shading-course/rad/s2015_pbs_rad_slides.pdf)[3] from the 2015 physically based shading course, it occurred to us that we should really package up this new testbed and release it alongside the presentation to serve as a working implementation of the concepts we were going to cover. But unfortunately this slipped through the cracks: the new testbed required a lot of work in order to make it useful, and both Dave and I were really pressed for time due to multiple new projects ramping up at the office.

Now, more than a year after our SIGGRAPH presentation, I’m happy to announce that we’ve finally produced and published a working code sample that demonstrates baking of Spherical Gaussian lightmaps! This new app, which I call “The Baking Lab”, is essentially a combination of the two testbed applications that we created. It includes all of the fun features that we were researching in 2015, but also includes real-time progressive baking of 2D lightmaps in various formats. It also allows switching to a progressive path tracer at any time, which serves as the “ground truth” for evaluating lightmap quality and accuracy. Since it’s an amalgamation of two older apps, it uses D3D11 and the older version of my sample framework. So there’s no D3D12 fanciness, but it will run on Windows 7. If you’re just interested in looking at the code or running the app, then go ahead and head over to GitHub: <https://github.com/TheRealMJP/BakingLab>. If you’re interested in the details of what’s implemented in the app, then keep reading.

**Lightmap Baking**

The primary feature of The Baking Lab is lightmap baking. Each of the test scenes includes a secondary UV set that contains non-overlapping UV’s used for mapping the lightmap onto the scene. Whenever the app starts or a new scene is selected, the baker uses the GPU to rasterize the scene into lightmap UV space. The pixel shader outputs interpolated vertex components like position, tangent frame, and UV’s to several render targets, which use MSAA to simulate conservative rasterization. Once the rasterization is completed, the results are copied back into CPU-accessible memory. The CPU then scans the render targets, and extracts “bake points” from all texels covered by the scene geometry. Each of these bake points represents the location of a single hemispherical probe to be baked.

Once all bake points are extracted, the baker begins running using a set of background threads on the CPU. Each thread continuously grabs a new work unit consisting of a group of contiguous bake points, and then loops over the bake points to compute the result for that probe. Each probe is computed by invoking a path tracer, which uses [Embree](https://embree.github.io/)[4] to allow for arbitrary ray tracing through the scene on the CPU. The path tracer returns the incoming radiance for a direction and starting point, where the radiance is the result of indirect lighting from various light sources as well as the direct lighting from the sky. The path tracer itself is a very simple unidirectional path tracer, using a few standard techniques like importance sampling, [correlated multi-jittered sampling](http://graphics.pixar.com/library/MultiJitteredSampling/paper.pdf)[5], and russian roulette to increase performance and/or convergence rates. The following baking modes are supported:

* **Diffuse** - a single RGB value containing the result of applying a standard diffuse BRDF to the incoming lighting, with an albedo of 1.0
* **Half-Life 2** - directional irradiance projected onto the [Half-Life 2 basis](http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_ShadingInValvesSourceEngine.pdf)[6], making for a total of 3 sets of RGB coefficients (9 floats total)
* **L1 SH** - radiance projected onto the first two orders of spherical harmonics, making for a total of 4 sets of RGB coefficients (12 floats total). Supports environment specular via a 3D lookup texture.
* **L2 SH** - radiance projected on the first three orders of spherical harmonics, making for a total of 9 sets of RGB coefficients (27 floats total). Supports environment specular via a 3D lookup texture.
* **L1 H-basis** - irradiance projected onto the first two orders of [H-basis](https://www.cg.tuwien.ac.at/research/publications/2010/Habel-2010-EIN/)[7], making for a total of 4 sets of RGB coefficients (12 floats total).
* **L2 H-basis** - irradiance projected onto the first three orders of [H-basis](https://www.cg.tuwien.ac.at/research/publications/2010/Habel-2010-EIN/), making for a total of 6 sets of RGB coefficients (18 floats total).
* **SG5** - radiance represented by the sum of 5 SG lobes with fixed directions and sharpness, making for a total of 5 sets of RGB coefficients (15 floats total). Supports environment specular via an approximate evaluation of per-lobe specular contribution.
* **SG6** - radiance represented by the sum of 6 SG lobes with fixed directions and sharpness, making for a total of 6 sets of RGB coefficients (18 floats total). Supports environment specular via an approximate evaluation of per-lobe specular contribution.
* **SG9** - radiance represented by the sum of 9 SG lobes with fixed directions and sharpness, making for a total of 9 sets of RGB coefficients (27 floats total). Supports environment specular via an approximate evaluation of per-lobe specular contribution.
* **SG12** - radiance represented by the sum of 12 SG lobes with fixed directions and sharpness, making for a total of 12 sets of RGB coefficients (36 floats total). Supports environment specular via an approximate evaluation of per-lobe specular contribution.

For SH, H-basis, and HL2 basis baking modes the path tracer is evaluated for random rays distributed about the hemisphere so that Monte Carlo integration can be used to integrate the radiance samples onto the corresponding basis functions. This allows for true progressive integration, where the baker makes N passes over each bake point, each time adding a new sample with the appropriate weighting. It looks pretty cool in action:

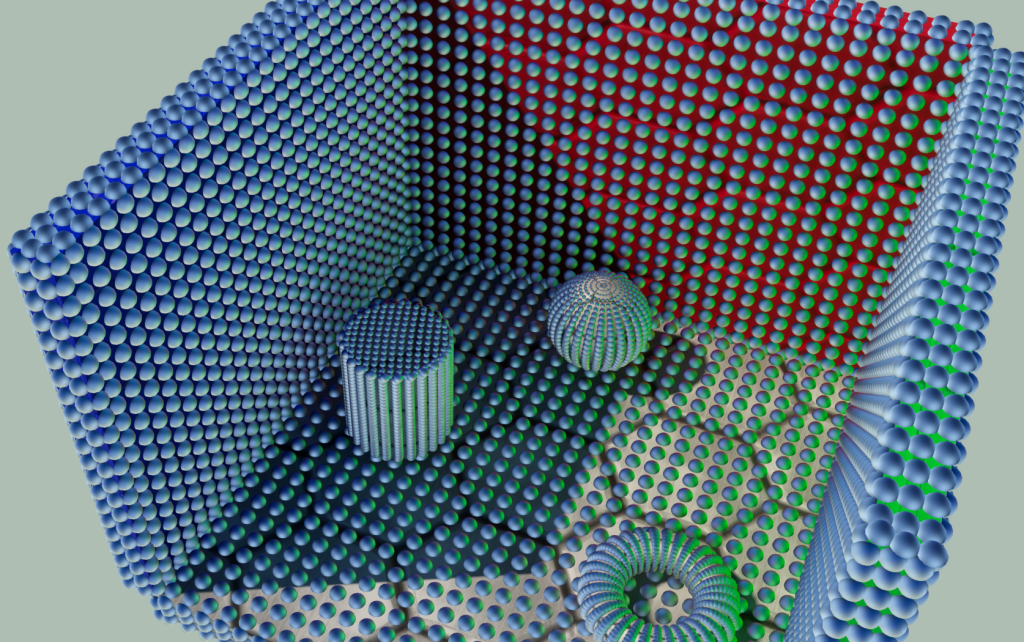
<https://youtu.be/aN0StrS2moI>

The same approach is used for the “Diffuse” baking mode, except that sampling rays are evaluated using a [cosine-weighted hemispherical sampling scheme](http://www.rorydriscoll.com/2009/01/07/better-sampling/)[8]. For SG baking, things get a little bit trickier. If the ad-hoc projection mode is selected, the result can be progressively evaluated in the same manner as the non-SG bake modes. However if either the Least Squares or Non-Negative Least Squares mode are active, we can’t run the solve unless we have all of the hemispherical radiance samples available to feed to the solver. In this case we switch to a different baking scheme where each thread fully computes the final value for every bake point that it operates on. However the thread only does this for a single bake point from each work group, and afterwards it fills in the rest of the neighboring bake points (which are arranged in a 8x8 group of texels) with the results it just computed. Each pass of of baker then fills in the next bake point in the work group, gradually computing the final result for all texels in the group. So instead of seeing the quality slowly improve across the light map, you see extrapolated results being filled in. It ends up looking like this:

<https://youtu.be/WhSlZgycQfM>

While it’s not as great as a true progressive bake, it’s still better than having no preview at all.

The app supports a few settings that control some of the bake parameters, such as the number of samples evaluated per-texel and the overall lightmap resolution. The “Scene” group in the UI also has a few settings that allow toggling different components of the final render, such as the direct or indirect lighting or the diffuse/specular components. Under the “Debug” setting you can also toggle a neat visualizer that shows a visual representation of the raw data stored in the lightmap. It looks like this:



**Ground Truth Path Tracer**

The integrated path tracer is primarily there so that you can see how close or far off you are when computing environment diffuse or specular from a light map. It was also a lot of fun to write - I recommend doing it sometime if you haven’t already! Just be careful: it may make you depressed to see how poorly your real-time approximation holds up when compared with a proper offline render. :-)

The ground truth renderer works in a similar vein to the lightmap baker: it kicks off multiple background threads that each grab work groups of 16x16 pixels that are contiguous in screen space. The renderer makes N passes over each pixel, where each pass adds an additional sample that’s weighted and summed with the previous results. This gives you a true progressive render, where the result starts out noisy and  (very) gradually converges towards a noise-free image:

<https://youtu.be/LUrCpZYQbm0>

The ground truth renderer is activated by checking the “Show Ground Truth” setting under the “Ground Truth” group. There’s a few more parameters in that group to control the behavior of the renderer, such as the number of samples used per-pixel and the scheme used for generating random samples.

**Light Sources**

There’s 3 different light sources supported in the app: a sun, a sky, and a spherical area light. For real-time rendering, the sun is handled as a directional light with an intensity computed automatically using the [Hosek-Wilkie solar radiance model](http://cgg.mff.cuni.cz/projects/SkylightModelling/)[9]. So as you change the position of the sun in the sky, you’ll see the color and intensity of the sunlight automatically change. To improve the real-time appearance, I used the disk area light approximation from the 2014 Frostbite presentation. The path tracer evaluates the sun as an infinitely-distant spherical area light with the appropriate angular radius, with uniform intensity and color also computed from the solar radiance model. Since the path tracer handles the sun as a true area light source, it produces correct specular reflections and soft shadows. In both cases the sun defaults to correct real-world intensities using actual photometric units. There is a parameter for adjusting the sun size, which will result in the sun being too bright or too dark if manipulated. However there’s another setting called “Normalize Sun Intensity” which will attempt to maintain roughly the same illumination regardless of the size, which allows for changing the sun appearance or shadow softness without changing the overall scene lighting.

The default sky mode (called “Procedural”) uses the Hosek-Wilkie sky model to compute a procedural sky from a few input parameters. These include turbidity, ground albedo, and the current sun position. Whenever the parameters are changed, the model is cached to a cubemap that’ s used for real-time rendering on the GPU. For CPU path tracing, the the sky model is directly evaluated for a direction using the sample code provided by the authors. When combined with the procedural sun model, the two light sources form a simple outdoor lighting environment that corresponds to real-world intensities. Several other sky modes are also supported for convenience. The “Simple"mode takes just a color and intensity as input parameter, and flood-fills the entire sky with a value equal to color \* intensity. The “Ennis”, “Grace Cathedral”, and “Uffizi Cross” modes use corresponding HDR environment maps to fill the sky instead of a procedural model.

For local lighting, the app supports enabling a single spherical area light using the “Enable Area Light” setting.  The area light can be positioned using the Position X/Position Y/Position Z settings, and its radius can be specified with the “Size” setting. There are a 4 different modes for specifying the intensity of the light:

* **Luminance** - the intensity corresponds to the amount of light being emitted from the light source along an infinitesimally small ray towards the viewer or receiving surface. Uses units of cd/m2. Changing the size of the light source will change the overall illumination the scene.
* **Illuminance** - specifies the amount of light incident on a surface at a set distance, which is specified using the “Illuminance Distance” setting. So instead of saying “how much light is coming out of the light source” like you do with the “Luminance” mode, you’re saying “how much diffuse light is being reflected from a perpendicular surface N units away”. Uses units of lux, which are equivalent to lm/m2.  Changing the size of the light source will  *not* change the overall illumination the scene.
* **Luminous Power** - specifies the total amount of light being emitted from the light source in all directions. Uses units of lumens. Changing the size of the light source will  *not* change the overall illumination the scene.
* **EV100** - this is an alternative way of specifying the luminance of the light source, using the [exposure value](https://en.wikipedia.org/wiki/Exposure_valuehttps:/en.wikipedia.org/wiki/Exposure_value)[10] system originally suggested by [Nathan Reed](http://www.reedbeta.com/blog/2014/06/04/artist-friendly-hdr-with-exposure-values/)[11]. The base-2 logarithmic scale for this mode is really nice, since incrementing by 1 means doubling the perceived brightness. Changing the size of the light source will change the overall illumination the scene.

The ground truth renderer will evaluate the area light as a true spherical light source, using importance sampling to reduce variance. The real-time renderer approximates the light source as a single SG, and generates very simple hard shadows using an array of 6 shadow maps. By default only indirect lighting from the area light will be baked into the lightmap, with the direct lighting evaluated on the GPU. However if the “Bake Direct Area Light” setting is enabled, then the direct contribution from the area light will be baked into the lightmap.

Note that all light sources in the app are always scaled down by a factor of 2-10 before being using in rendering, as suggested by Nathan Reed in [his blog post](http://www.reedbeta.com/blog/2014/06/04/artist-friendly-hdr-with-exposure-values/)[11]. Doing this effectively shifts the window of values that can be represented in a 16-bit floating point value, which is necessary in order to represent specular reflections from the sun. However the UI always will always show the unshifted values, as will the debug luminance picker that shows the final color and intensity of any pixel on the screen.

**Exposure and Depth of Field**

As I mentioned earlier, the app implements a physically based exposure system that attempts to models the behavior and parameters of a real-world camera. Much of the implementation was based on the code from Padraic Hennessy’s [excellent series of articles](https://placeholderart.wordpress.com/2014/11/16/implementing-a-physically-based-camera-understanding-exposure/)[12], which was in turn inspired by Sébastien Lagarde and Charles de Rousiers’s [SIGGRAPH presentation from 2014](http://www.frostbite.com/wp-content/uploads/2014/11/course_notes_moving_frostbite_to_pbr_v2.pdf)[2]. When the “Exposure Mode” setting is set to the “Manual (SBS)” or “Manual (SOS)” modes, the final exposure value applied before tone mapping will be computed based on the combination of aperture size, ISO rating, and shutter speed. There is also a “Manual (Simple)” mode available where a single value on a log2 scale can be used instead of the 3 camera parameters.

Mostly for fun, I integrated a post-process depth of field effect that uses the same camera parameters (along with focal length and film size) to compute per-pixel circle of confusion sizes. The effect is off by default, and can be toggled on using the “Enable DOF” setting. Polygonal and circular bokeh shapes are supported using the technique suggested by Tiago Sousa in his [2013 SIGGRAPH presentation](http://advances.realtimerendering.com/s2013/Sousa_Graphics_Gems_CryENGINE3.pptx)[13]. Depth of field is also implemented in the ground truth renderer, which is capable of achieving true multi-layer effects by virtue of using a ray tracer.

[](https://therealmjp.github.io/images/converted/sg-series-part-6-step-into-the-baking-lab/dof_gt.png)

**Tone Mapping**

Several tone mapping operators are available for experimentation:

* **Linear** - no tone mapping, just a clamp to [0, 1]
* **Film Stock** - [Jim Hejl](https://twitter.com/jimhejl) and Richard Burgess-Dawson’s polyomial approximation of [Haarm-Peter Duiker](https://twitter.com/hpduiker)’s filmic curve, which was created by scanning actual film stock. Based on the implementation provided by [John Hable](http://filmicgames.com/archives/75)[14].
* **Hable (Uncharted2)** - [John Hable](https://twitter.com/FilmicWorlds)’s adjustable filmic curve from his [GDC 2010 presentation](http://www.gdcvault.com/play/1012351/Uncharted-2-HDR)[15]
* **Hejl 2015** - Jim Hejl’s filmic curve that he [posted on Twitter](https://twitter.com/jimhejl/status/633777619998130176)[16], which is a refinement of Duiker’s curve
* **ACES sRGB Monitor** - a fitted polynomial version of the [ACES](https://github.com/ampas/aces-dev)[17] reference rendering transform (RRT) combined with the sRGB monitor output display transform (ODT), generously provided by [Stephen Hill](https://twitter.com/self_shadow).

**Debug Settings**

At the bottom of the settings UI are a group of debug options that can be selected. I already mentioned the bake data visualizer previously, but it’s worth mentioning again because it’s really cool. There’s also a “luminance picker”, which will enable a text output showing you the luminance and illuminance of the surface under the mouse cursor. This was handy for validating the physically based sun and sky model, since I could use the picker to make sure that the lighting values matched what you would expect from real-world conditions. The “View Indirect Specular” option causes both the real-time renderer and the ground truth renderer to only show the indirect specular component, which can be useful for gauging the accuracy of specular computed from the lightmap. After that there’s a pair of buttons for saving or loading light settings. This will serialize the settings that control the lighting environment (sun direction, sky mode, area light position, etc.) to a file, which can be loaded in whenever you like. The “Save EXR Screenshot” is fairly self-explanatory: it lets you save a screenshot to an EXR file that retains the HDR data. Finally there’s an option to show the current sun intensity that’s used for the real-time directional light.

**References**

[1] [Academy Color Encoding System](https://en.wikipedia.org/wiki/Academy_Color_Encoding_System)  
[2] [Moving Frostbite to PBR (course notes)](http://www.frostbite.com/wp-content/uploads/2014/11/course_notes_moving_frostbite_to_pbr_v2.pdf)  
[3] [Advanced Lighting R&D at Ready At Dawn Studios](http://blog.selfshadow.com/publications/s2015-shading-course/rad/s2015_pbs_rad_slides.pdf)  
[4] [Embree: High Performance Ray Tracing Kernels](https://embree.github.io/)  
[5] [Correlated Multi-Jittered Sampling](http://graphics.pixar.com/library/MultiJitteredSampling/paper.pdf)  
[6] [Shading in Valve’s Source Engine](http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_ShadingInValvesSourceEngine.pdf)  
[7] [Efficient Irradiance Normal Mapping](https://www.cg.tuwien.ac.at/research/publications/2010/Habel-2010-EIN/)  
[8] [Better Sampling](http://www.rorydriscoll.com/2009/01/07/better-sampling/)  
[9] [Adding a Solar Radiance Function to the Hosek Skylight Model](http://cgg.mff.cuni.cz/projects/SkylightModelling/)  
[10] [Exposure value](https://en.wikipedia.org/wiki/Exposure_value)  
[11] [Artist-Friendly HDR With Exposure Values](http://www.reedbeta.com/blog/2014/06/04/artist-friendly-hdr-with-exposure-values/)  
[12] [Implementing a Physically Based Camera: Understanding Exposure](https://placeholderart.wordpress.com/2014/11/16/implementing-a-physically-based-camera-understanding-exposure/)  
[13] [CryENGINE 3 Graphics Gems](http://advances.realtimerendering.com/s2013/Sousa_Graphics_Gems_CryENGINE3.pptx)  
[14] [Filmic Tonemapping Operators](http://filmicgames.com/archives/75)  
[15] [Uncharted 2: HDR Lighting](http://www.gdcvault.com/play/1012351/Uncharted-2-HDR)  
[16] [Jim Hejl on Twitter](https://twitter.com/jimhejl/status/633777619998130176)  
[17] [Academy Color Encoding System Developer Resources](https://github.com/ampas/aces-dev)